base latency for the transmission of a message of L bits along D channels. Make the analogy between packets without headers and flits in wormhole switching, and determine the optimal flit size.

2.8 Consider a network using wormhole switching, and a node architecture that is able to send and receive up to four packets simultaneously without interference. The start-up latency to initiate a new packet transmission after initiating the previous one is t_s. Assuming that there are four minimal paths between two nodes that are D links apart, compute the base latency to send a message of L bits when the message is split into four sequences of packets, each sequence following a different path. Assume that routers are input-buffered only.

2.9 A hierarchical network topology has channels of two different widths, W and $\frac{W}{2}$, all of them being clocked at the same frequency B. The network uses wormhole switching, and routers have input buffers and output buffers deep enough to avoid filling the buffers in the absence of contention. Internal data paths are W bits wide. Compute the base latency to send a message of L bits across two channels in two cases: (a) the width of the first and second channels are W and $\frac{W}{2}$, respectively; (b) the width of the first and second channels are $\frac{W}{2}$ and W, respectively. Assume that messages are not split into packets.

Chapter 7

Network Architectures

In many instances, particularly in fine-grained parallel machines, the performance of the applications are communication-limited rather than computation-limited. In this case, the design of the network architecture is crucial. By network architecture we refer to the topology of interconnection between the processing nodes, and the data and control path of the routers within the individual nodes. The design of the network architecture is fundamentally a process of making trade-offs between performance parameters such cost, latency, and throughput, and physical constraints such as area, wireability, and I/O limitations. Since physical constraints are continually evolving with advances in fabrication and packaging technology, so too are the appropriate choices of network topology and design alternatives for the router architecture. This chapter discusses basic issues facing the choice of network topologies and presents the design of router architectures that have been successfully employed within these topologies.

Chapter 1 describes many distinct topologies that have been proposed for use in multiprocessor architectures. The majority of these topologies are either configurations of, or isomorphic to, k-ary n-cube networks. These include the binary hypercube, tori, rings, and meshes. Indirect networks such as the indirect binary n-cube and Omega networks are also topologically closely related to these networks. Low dimensional k-ary n-cube networks possess desirable features with respect to their physical implementations, such as constant degree and ease of embedding in physical space. They also possess desirable properties with respect to message performance such as simple distributed routing algorithms. As a result, these networks have been used in the majority of commercial systems developed in the past decade. These include the iPSC series of machines from Intel, the Cray T3D and T3E, the Ametek 2010, MasPar MP-1 and MP-2, and Thinking Machines CM-1 and CM-2. These networks have formed the communications substrate of many commercial machines and are therefore the focus of this chapter. We observed that the network topology is primarily influenced by the packaging technology which places constraints on the wiring density, chip pin-out, and wire lengths. In the presence of these constraints, topological design issues must reconcile conflicting demands in choices of channel width, network dimensionality, and wire length. Thus, this chapter initially focuses on the optimization of the network topology.

While the architecture of the routers is certainly related to the topology of the interconnect, their design is primarily influenced by the switching technique that they are designed to support. Generic issues can be broadly classified into those dealing with internal data and control paths and those dealing with the design of the interrouter physical links. Specific design choices include internal switch design, buffer management, use of virtual channels, and physical channel flow control strategies. The remainder of the chapter presents descriptions of several modern router architectures as examples of specific design choices. Rather than present a comprehensive coverage of available architectures, we have tried to select illustrative examples from the range of currently available architectures.

7.1 Network Topology and Physical Constraints

For a fixed number of nodes, the choice of the number of dimensions of a k-ary n-cube represents a fundamental trade-off between network diameter and node degree. This choice of network dimension also places different demands on physical resources such as wiring area and number of chip I/Os. For a given implementation technology, practical constraints on these physical resources will determine architectural features such as channel widths, and as a result determine the *no-load message latency*: message latency in the absence of traffic. Similarly, these constraints will also determine the degree of congestion in the network for a given communication pattern, although accurate modeling of such dynamic behavior is more difficult. It thus becomes important to model the relationships between physical constraints and topology, and the resulting impact on performance. Network optimization is the process of utilizing these models in selecting topologies that best match the physical constraints of the implementation. The following subsections discuss the dominant constraints and the construction of analytic models that incorporate their effects on the no-load message latency.

7.1.1 Bisection Bandwidth Constraints

One of the physical constraints facing the implementation of interconnection networks is the available wiring area. The available wiring area is determined by the packaging technology, i.e., does the network reside on a chip, multichip module or printed circuit board. In particular, VLSI systems are generally wire-limited: the silicon area required by these systems is determined by interconnect area and the performance is limited by the delay of these interconnections. Since these networks must be implemented in three dimensions, for an N node network the available wiring space grows as $N^{\frac{2}{3}}$ while the network traffic can grow as N. Clearly machines cannot be scaled to arbitrarily large sizes without eventually encountering wiring limits. The choice of network dimension is influenced by how well the resulting topology makes use of available wiring area. Such an analysis requires performance measures that can relate network topology to the wiring constraints.

On such performance measure is the *bisection width* of a network: the minimum number of wires that must be cut when the network is divided into two equal sets of nodes. Since the primary goal is one of assessing bandwidth and resulting wiring demands, only wires carrying data are generally counted, excluding control, power, and ground signals. However, these effects are relatively easy to incorporate in the following analysis. The collective bandwidth over these wires is referred to as the *bisection bandwidth*. For example, consider a 2-D torus with $N = k^2$ nodes. Assume that each pair of adjacent nodes are connected with one bidirectional data channel of width W bits. If this network is divided into two halves, there will be $2W\sqrt{N}$ wires crossing this bisection. The factor of 2 is due to the wraparound channels. Imagine what happens when we bisect a 3-D torus with $N = k^3$ nodes. Each half comprises of $\frac{k^3}{2}$ nodes. A 2-D plane of k^2 nodes has links crossing the bisection, leading to a bisection width of $2Wk^2 = 2Wk^{n-1}$. These cases are illustrated in Figure 7.1. For the sake of clarity the wraparound links in the 3-D topology are not shown. In general, a bisection of a k-ary n-cube will cut channels from a k-ary $(n-1)$-cube leading to a bisection width of $2Wk^{n-1}$.

Using the above approach we can compute the bisection width of a number of popular topologies. These are summarized in Table 7.1. The implicit assumption here is that all of these topologies are regular, i.e., each dimension has the same radix. Consider the case of networks with $N =$

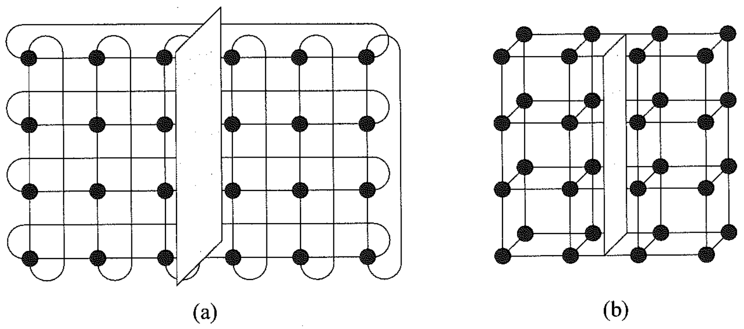


Figure 7.1. Examples of the computation of bisection width: (a) 2-D torus. (b) 3-D torus (wraparound links not shown).

$k_1 \times k_2 \times \ldots \times k_n$ nodes. In this case the minimum bisection width is orthogonal to the dimension with the largest radix. For example, this is the case shown in Figure 7.1. If the largest radix is k_m, then we have $\frac{N}{k_m}$ nodes with channels across this bisection. Thus, the bisection width can be expressed as $\frac{2WN}{k_m}$ bits.

The above computation assumes that each node devotes W pins to communication with a neighbor along a dimension. No assumptions are made about how these pins are used. In general, given a set of W pins between adjacent nodes, these may be utilized as two (opposite) $\frac{W}{2}$ bit unidirectional channels, one W-bit bidirectional channel, or one W-bit unidirectional channel. In the computing bisection width, the manner is which these channels are used is not particularly important. However, it is central to the evaluation of performance since channel widths directly impact message latency. We will see in later sections that assumptions about how these pins are used must be unambiguous and clearly stated.

7.1.2 Node Size Constraints

In addition to wiring space, a second physical constraint is the number of I/Os available per router. This is referred to as the *node size.* The construction of networks under a constant channel width for an arbitrary number of dimensions is impractical since the node size is linear in the number of dimensions. For example, consider a 2-D network and assume a baseline channel width of 16 data

Table 7.1. Examples of bisection width and node size of some common networks ($t \times t$ switches are assumed in the Omega network).

Network	Bisection Width	Node Size
k-ary n-cube	$2Wk^{n-1}$	$2Wn$
Binary n-cube	$\frac{NW}{2}$	nW
n-dimensional mesh	Wk^{n-1}	$2Wn$
Omega network	NW	$2tW$

bits, 8 control bits, and full-duplex channels, i.e., two unidirectional channels between adjacent nodes. This would correspond to 192 signal pins not including power, ground, and some miscellaneous I/Os. This is certainly feasible in current technology. For an n-dimensional network, the router will have 48 pins in each direction in each dimension for a total pin-out of $96n$. If we consider a 10-D network with the same channel width, we will require 960 pins, or close to 500 pins just for the channels in a more modest 5-D network. As we progress to 32-bit channels, the feasible network dimension becomes smaller. Thus, practical considerations place limits on channel width as a function of dimensionality [1, 3].

Even if we consider the bisection width constraints and networks being wiring area limited, it is likely that before networks encounter a bisection width limit, they may encounter the pin-out limit. This is evident from the following observation. Consider the implementation of a k-ary n-cube with the bisection width constrained to be N bits. The maximum channel width determined by this bisection width is given by

$$\text{Bisection width} = 2Wk^{n-1} = N \Rightarrow W = \frac{k}{2} \tag{7.1}$$

Note that this is the maximum channel width permitted under this bisection width constraint. A router does not have to provide this number of pins across each channel. However, if it does, the router will have a channel width of W bits between adjacent routers in each direction in each dimension for a total pin-out of $2nW = nk$. A network of 2^{20} nodes will have a pin-out of 2,048 in a 2-D configuration and a pin-out of 128 in a 4-D configuration. The former is certainly infeasible in 1997 technology for commercial single-chip packaging. In general, both the maximum number of available pins and the wiring area constraint as represented by the bisection width place upper bounds on the channel width. If the maximum number of pins available for data channels is given by P, and the bisection width is given by B, then the channel width is constrained by the following two relations. The feasible channel width is the smaller of the two.

$$W \leq \frac{P}{2n}$$
$$W \leq \frac{B}{2k^{n-1}} \tag{7.2}$$

To understand the effect of channel width on message latency consider the following. For a fixed pin-out, higher-dimensional networks will have smaller channel widths: a fixed number of pins partitioned across a higher number of dimensions. As we reduce the number of dimensions, the channel width increases. For example, if the channel width of an n-dimensional binary hypercube is W_h, then the total number of pins is nW_h (note the absence of the factor of 2 since $k = 2$ is special case with no wraparound channels). If we keep this pin-out constant, then for an m-dimensional torus with k_1^m nodes, the pin-out is $2mW_m$. The channel width of the torus is given by

$$W_m = W_h \times \frac{n}{2} \times \frac{1}{m} \tag{7.3}$$

However, if we keep the number of nodes constant, we have

$$N = 2^n = k_1^m \Rightarrow k_1 = N^{\frac{1}{m}} \tag{7.4}$$

Therefore we see that as the dimension decreases, the linear increase in channel width is offset by an exponential increase in the distance traveled by a message in each dimension. Expressions for message latency must capture these effects in enabling the analysis of trade-offs between bisection width, pin-out, and wiring length.

7.1.3 Wire Delay Constraints

As pointed out in [71], in VLSI systems the cycle time of the network is determined by the maximum wire delay, while the majority of the power is expended in driving these wires. Thus, topologies that can be embedded in two and three dimensions with short wires will be preferred. While even low-dimensional networks may logically appear to have long wires due to the wraparound connections, these can be avoided by interleaving nodes as shown in Figure 7.2 [258, 312]. However, when higher-dimensional networks are embedded in two and three dimensions, longer long wires do result. For the purposes of analyzing the effect of wire length we can determine this increase in wire length as follows [3]. The following analysis is based on an embedding in two dimensions, although it can be extended to three dimensions in a straightforward manner.

A k-ary n-cube is embedded in two dimensions by embedding $\frac{n}{2}$ dimensions of the network into each of the two physical dimensions (assuming an even number of dimensions). After embedding the first two dimensions each additional dimension increases the number of nodes in the network by a factor of k. Embedding in two dimensions provides an increase in the number nodes in each physical dimension by a factor of $\sqrt{k}$. If we ignore wire width and wiring density effects, the length of the longest wire in each physical dimension is also increased by a factor of $\sqrt{k}$. Note that if we had to account for the thickness of the wires, the length of the longest wire would actually grow faster than $\sqrt{k}$ as we increase the number of dimensions (see [3] for a good discussion). Ignoring wire width effects, the length of the longest wire in a 2-D embedding of an n-dimensional network increases by a factor of $k^{\frac{n-2}{2}}$ over the maximum wire length of a 2-D network. For a generalization to embedding in three dimensions and higher, refer to Exercise 7.3.

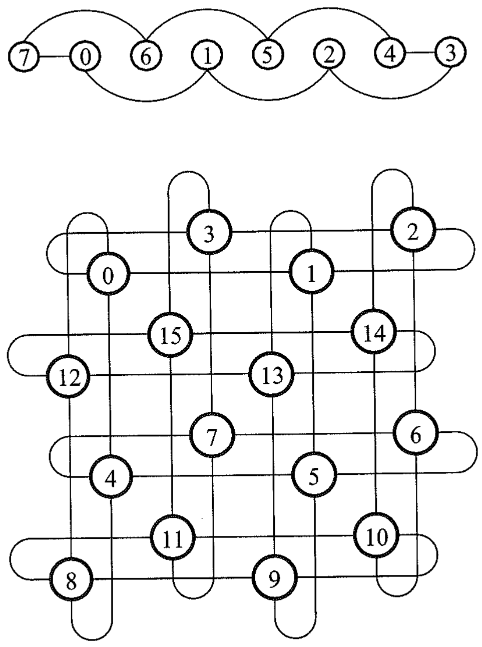


Figure 7.2. Interleaving nodes to avoid the long wire lengths due to the wraparound connections.

The delay along the wire may follow one of several models depending upon the length, the implementation medium, e.g., aluminum on silicon, copper on polyimide, etc. The three common wire delay models include the linear, logarithmic, and constant delay models.

Delay $\propto \log_e l$	Logarithmic delay model
Delay $\propto l$	Linear delay model
Delay $\propto C$	Constant delay model

where l is the wire length. These delay models are incorporated into a model of average message latency developed in the following section.

7.1.4 A Latency Model

Given the impact of the preceding physical constraints, we can now develop a model of the no-load latency of a message in a k-ary n-cube in the presence of these constraints. Such relationships between performance and implementation constraints help us make informed trade-offs in the choice of network topology. In the following we assume a wormhole-switched network, and the development follows that provided in [3].

Consider the base latency of a wormhole-switched message from Equation 2.4.

$$t_{wormhole} = D(t_r + t_s + t_w) + \max(t_s, t_w)\left\lceil \frac{L}{W} \right\rceil \tag{7.5}$$

where t_r is the time to make a routing decision, t_s is the delay through the switch, and t_w is the wire delay across the physical channel. The first term in the right hand side of the expression represents the delay due to the distance between the source and destination. The major components are the routing, switch, and link delays. The routing protocols discussed in Chapter 4 implement decisions that can be quite complex relative to the no-load delay of a data flit through the switch. While the impact of this routing delay is reduced for long messages, it may be nonnegligible for small messages, and is represented by t_r. It is sometimes the case that the switch delays (t_s) are greater than the link delays, and as a result have a nonnegligible impact on the performance of the network. The development of the latency model in this section assumes switch delays dominate wire delays and follows the approach in [3]. Later in this chapter we consider the case where wire delays dominate switch delays. Finally, t_w represents the wire delay across the physical link. The above expression is based on a router architecture where the inputs and outputs are buffered. Thus, once a message pipeline has been set up, the period of this pipeline is determined by the greater of the switch delay and wire delay. If we had used an input only buffered switch, then $\max(t_s, t_w)$ would have simply been replaced by $(t_s + t_w)$.

Under random traffic, the value of the distance D above can be replaced by the average distance between any pair of nodes. With bidirectional links, in an n-dimensional network a message will travel an average of $\frac{k}{4}$ links in a dimension when k is even and $\frac{(k^2-1)}{4k}$ links when k is odd. To capture the effect of wire delays the routing, switching, and wire delays can be normalized to the wire delay between adjacent nodes in a 2-D network. This will permit analysis that would remain relevant with improvements in technology. As new packaging techniques, materials, and interconnect media evolve, wire delays between adjacent nodes in a 2-D embedding will improve. The latency expression using normalized delays will represent performance relative to this improved base 2-D case. Let the switch delay and routing delay be represented as integer multiples of the wire delay in a 2-D network: s and r, respectively. From the earlier discussion, we know the value of t_w relative to a 2-D network. Thus, the latency expression can be written as follows (for networks where k is even):

$$t_{wormhole} = n\frac{k}{4}(r + s + k^{\frac{n}{2}-1}) + \max(s, k^{\frac{n}{2}-1}) \left\lceil \frac{L}{W} \right\rceil \tag{7.6}$$

The above expression is based on a linear delay model for wires. For short wires, delay is logarithmic [71] and for comparison purposes, it is useful (though perhaps unrealistic) to have model based on constant wire delay. The above expression is normalized to the wire delay in a 2-D mesh. Therefore, in the constant delay model, wire delay simply remains at 1. In the logarithmic delay model, delay is a logarithmic function of wire length, i.e., $1 + \log_e l$ [71]. The latency expression under the logarithmic wire delay model can be written as

$$t_{wormhole} = n\frac{k}{4}\left[r + s + 1 + \left(\frac{n}{2} - 1\right)\log_e k\right] + \max\left[s, 1 + \left(\frac{n}{2} - 1\right)\log_e k\right]\left\lceil \frac{L}{W} \right\rceil \tag{7.7}$$

For the constant wire delay model the latency expression is

$$t_{wormhole} = n\frac{k}{4}(r + s + 1) + s\left\lceil \frac{L}{W} \right\rceil \tag{7.8}$$

The latency expressions can be viewed as the sum of two components: a *distance component* which is the delay experienced by the header flit, and a *message component*, which is the delay due to the length of the message. The expressions also include terms representing physical constraints (wire length, channel widths), network topology (dimensions, average distance), applications (message lengths) and router architecture (routing and switching delays). Although in the above expression routing and switching delays are constants, we could model them as an increasing function of network dimension. This appeals to intuition since a larger number of dimensions may make routing decisions more complex, and increase the delay through internal networks that must switch between a larger number of router inputs and outputs. We revisit this issue in the exercises at the end of this chapter.

The above expression provides us with insight into selecting the appropriate number of dimensions for a fixed number of nodes. The minimum latency is realized when the component due to distance is equal to the component due message length. The choice of parameters such as the switching and routing delays, channel widths, and message lengths determine the dimension at which this minimum is realized. The parameters themselves are not independent and are related by implementation constraints. Examples of such analysis are presented in the following sections.

7.1.5 Analysis

In this section we focus on the selection of the optimal dimension given a fixed number of nodes. The optimal dimension realizes the minimal value of the no-load latency. The choice of the optimal dimension is dependent on the technological parameters. Wiring density, pin-out, and wire delays each individually define a dimension that minimizes the no-load latency performance of the network. The following analysis is intended to demonstrate how the topological features of the network are dependent on constraints on each these three physical features, and thus enable us to select the topology that optimizes the performance of an implementation.

Constant Bisection Width

We first consider the case where the wiring area is limited, and this limited wiring resource is represented by a fixed bisection width. For the purposes of this analysis, let us assume this fixed resource is assumed to be equivalent to that of a k-ary n-cube with $k = 2$, and single-bit channels between adjacent nodes. Substituting $k = 2$ and $W = 1$ in the expression in Table 7.1, we obtain this constant bisection width as N. Note that this value differs from the expression for the special case of the n-dimensional binary hypercube in that wrap-around channels are assumed. The following analysis could just as easily be performed with with any known bisection width. From Section 7.1.2, we know that the corresponding channel width that fully utilizes this bisection width is $\frac{k}{2}$. Thus, the latency expression under the constant bisection width constraint can be obtained by substituting for W and can be written as

$$t_{wormhole} = n\frac{k}{4}(r + s + k^{\frac{n}{2}-1}) + \max(s, k^{\frac{n}{2}-1})\left\lceil \frac{2L}{k} \right\rceil \qquad (7.9)$$

In the above latency expression we implicitly assume that all of the W signals devoted to the channel between a pair of routers is used for message transmission, i.e., half-duplex channels. We could have assumed that the W signals were organized as two unidirectional physical channels — one in each direction across the channel. In this case the channel width in the above expression would have been replaced by $\frac{k}{4}$ rather than $\frac{k}{2}$. This illustrates the necessity of being careful about what exactly a W-bit channel refers to in order to maintain fair comparisons between networks. Comparing W-bit bidirectional channels and W-bit, dual, unidirectional channels is clearly not a fair comparison. Although both instances are sometimes described as W-bit channels between adjacent nodes, the latter has twice as many signals as the former. From the perspective of packaging and wiring area demands, we are simply interested in the number of pins devoted to communication with a neighbor in a dimension. A fair comparison between two networks would ensure the same number of signals devoted to communication between neighboring nodes in a dimension.

As we increase the number of dimensions, when the network is laid out in the plane (or even in three dimensions for that matter), the number of interprocessor channels that cross the bisection increases. If this bisection width is held constant, then the individual channel width must decrease, effectively making messages longer and increasing the component of message latency due to message length. The length of the longest wire also increases with increasing dimension increasing the distance component of the latency expression for higher dimensions. For a constant number of nodes, increasing the dimension increases the wire length by a factor of $N^{\frac{1}{(n+1)n}}$. The cumulative effect of all of these trade-offs is illustrated in Figure 7.3. It is apparent that for larger-size systems, the distance component of message latency dominates at low dimensions, thus favoring networks of dimension 3 or less. Smaller message sizes also effectively increase the impact of switching and routing delays. Thus, increasing r, s, and reducing L for the larger size systems (e.g., 2^{20} and 2^{16} nodes) will further increase the optimal dimension to 4 but not much further. For smaller systems (i.e., 2^{8}) the optimal number of dimensions remains at three since the latency in these systems is less dependent on the distance component. This behavior appeals to intuition since messages must travel through a greater number of routers in larger systems and thus would be more sensitive to switching and routing delays. Note that the large values of latency are due to the model component that provides an exponential penalty for wire length at high dimensions.

In general, we find that the trade-offs are a bit different for a larger number of nodes versus a smaller number of nodes. Larger systems are more sensitive to switch and wire delays and when these are large, can encourage the use of a larger number of dimensions, e.g., three to six. Smaller message sizes also have the effect of increasing the impact of switch and routing delays. Smaller

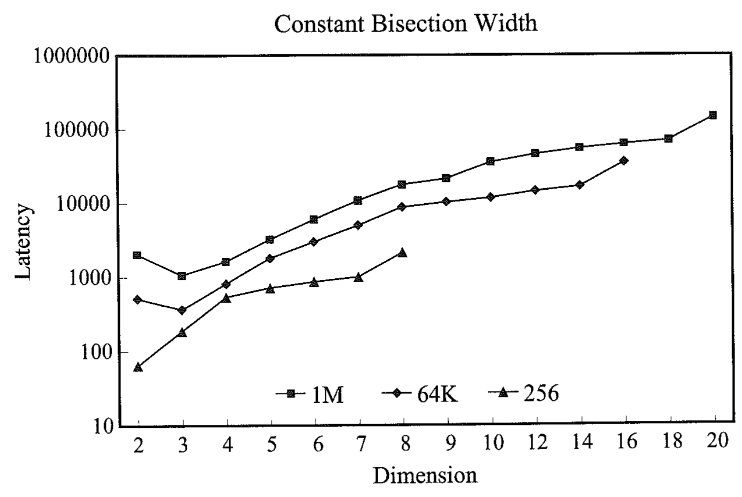


Figure 7.3. Effect of constant bisection width on the no-load message latency.

size systems are less sensitive to these delays. Rather, the relative impact of narrower channels has a greater effect on message latency than the switching and routing delays. The optimal number of dimensions for smaller systems tends to be about two to three. Finally, the use of linear wire delay models penalizes higher-dimension networks. However, use of the logarithmic wire delay models does not change the results appreciably. The optimal number of dimensions for large systems (> 1K) ranges from three to six, and for small systems (< 512) remains at two for realistic wire delay models. For small systems, the (unrealistic) constant delay model and small message sizes increases the optimal number of dimensions. However for the case of 16K and 1M nodes, the optimal number of dimensions rises to 9. These results are summarized in Tables 7.2a and 7.2b.

Finally we note that the behavior is dependent on two additional factors. The latency expressions are based on the no-load latency. The presence of link congestion and blocking within the network can produce markedly different behavior, particularly if the message destinations and lengths are nonuniform. Reference [3] presents one approach to incorporating models of network congestion into the latency expressions. A second issue is the design of the routers. If routers employ input

Table 7.2. Optimal number of dimensions for constant bisection bandwidth and: (a) Message size of 32 bits. (b) Message size of 256 bits.

(a)

Wire Delay Model	1M	16K	256
Linear	3	3	2
Logarithmic	7	6	2
Constant	9	7	3

(b)

Wire Delay Model	1M	16K	256
Linear	3	3	2
Logarithmic	4	3	2
Constant	6	4	2

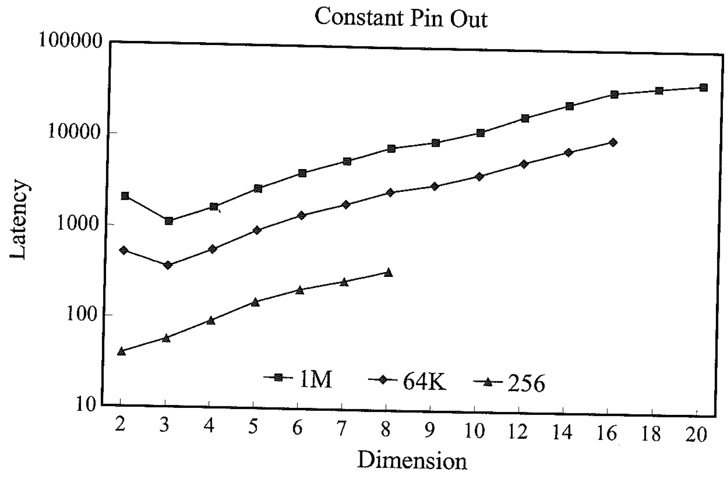


Figure 7.4. Effect of constant pin-out on the no-load message latency.

buffering, the cycle time of the message pipeline is the sum of the switch (t_s) and wire (t_w) delays. When both input and output buffering is employed, the cycle time is reduced to max (t_s, t_w). Thus, the latter model would favor higher dimensions since it is less dependent on wire delays particularly when switching delays are relatively large.

Constant Node Size

The preceding analysis can lead to designs where the individual nodes must have extremely large pin-out (often impractical in 1996–1997 technology) to make full use of the bisection bandwidth. In these instances network performance is limited not by the wiring resource but chip I/O or pin resources [1]. As we did for bisection width, we can re-write the latency expression to capture the dependency of latency on the network dimension assuming the number of chip I/Os are fixed. The following expression is written assuming linear wire delay and P pins per chip devoted to communication channels. Each router is assumed to communicate over W signals with an adjacent router in each direction in each dimension. Thus $P = 2nW$ and substituting $W = \frac{P}{2n}$ the no-load latency expression using a linear wire delay model becomes

$$t_{wormhole} = n\frac{k}{4}(r + s + k^{\frac{n}{2}-1}) + \max(s, k^{\frac{n}{2}-1}) \left\lceil \frac{2nL}{P} \right\rceil \tag{7.10}$$

The factor of 2 comes from the availability of channels in each direction within a dimension, and the use of W signals between adjacent nodes. For binary hypercubes we would have $W = \frac{P}{n}$ since there is only one direction in each dimension. A plot illustrating the optimal number of dimensions is shown in Figure 7.4 for the linear wire delay model provided in the above equation, parameterized as shown in the figure. The message size is fixed to that of a nominal-sized cache line in a shared-memory system, or medium size message (32 bytes) and we assume a nominal number of pins devoted to communication (256 pins). The optimal number of dimensions for large numbers of nodes with message size of 256 bits is 3, whereas for the smaller-sized system it is 2. If the wire delay model is

Table 7.3. Optimal number of dimensions for constant pin-out and: (a) Message size of 32 bits. (b) Message size of 256 bits.

(a)

Wire Delay Model	1M	16K	256
Linear	3	3	3
Logarithmic	9	7	4
Constant	9	8	4

(b)

Wire Delay Model	1M	16K	256
Linear	3	3	2
Logarithmic	6	4	2
Constant	9	7	3

changed to the logarithmic delay model the optimal number of dimensions increases to 6, 4, and 2 for the 1M, 16K, and 256-node systems respectively. Reducing the message size has a substantial effect further increasing the optimal number of dimensions. These results are summarized in Tables 7.3a and 7.3b.

The exponential increase in the distance in each dimension overshadows the linear increase in channel width as the network dimension is reduced. Under the constant node size constraint model, higher dimensionality is more important than wider channels [1]. The use of input and output buffering further encourages the use of higher dimensions. This model is also particularly sensitive to wire delays as shown in Figure 7.4.

This analysis is very dependent upon the structure of the physical channel. The model assumes that the number of pins devoted to communication with a neighboring node in a dimension is organized as a single bidirectional channel of width W bits. The channel width is a major determinant of message latency. The pins could have been organized as two unidirectional channels of width $\frac{W}{2}$ bits each, significantly altering the impact of message size. Our expectation is that with the use of bidirectional channels, the network would be relatively more sensitive to traffic patterns and load since messages can only cross the physical channel in one direction at a time. Such conflicts may be reduced by using full-duplex channels, or making the channels unidirectional. In the latter case the average distance a message travels within a dimensions is doubled (to $\frac{k-1}{2}$), and overall average message distance would be increased. The preceding analysis could be easily repeated for any of these relevant cases with appropriate adjustments to the expression that relates channel widths to the number of available data pins. The important point to note is that when pin resources are limited, they can be used in different ways with consequently differing impact on performance.

It is also important to note that while the analysis has been under a single constraint, the bisection width and node size are not independent. A fixed node size and dimensionality will produce a fixed bisection width. Using the model of the physical channel between adjacent routers as described in the preceding paragraphs, we observe that the channel width $W = \frac{P}{2n}$. Based on this channel width, the relationship between node size and bisection width can be written from Table 7.1 as

$$Bisection\ width = \frac{P}{n}\ k^{n-1} \tag{7.11}$$

Bisection width is linearly related to node size and inversely related to the network dimensionality. As networks are scaled to larger sizes, the technology-dependent question is one of whether bisection width or node size limits are encountered first. The answer is based on a moving target, and may be different at any given point in the technology curve.

Constant Wire Throughput

The analysis described in the preceding sections were based on the assumption that switching delays were dominant. Therefore the latency model represented these delays (and routing delays) as values normalized to the wire delay of between adjacent routers in a 2-D implementation. As the speed of operation continues to rise and systems employ larger networks, wire delays begin to dominate. This section develops the latency model in the case where it is wire delays that are dominant rather than switching delays.

When the delay along the wire is small relative to the speed of transmission through the wire media and propagation delay through the drivers, the lines can be modeled as simple capacitive loads. This is the case for lower-dimensional networks. When higher-dimensional networks are laid out in 2 or 3 dimensions, the propagation delay along the longest wire can be substantial. In reality, a signal connection actually operates as a transmission line. As the speeds of network operation increase, and high-dimensional networks are mapped to two and three dimensions producing longer wires, a more accurate analysis utilizes models of physical channel behavior based on transmission lines. Consider the following analysis [31]. A signal propagates along a wire at a speed given by the following expression.

$$v = \frac{c_0}{\sqrt{\epsilon_r}} \tag{7.12}$$

The value of c_0 is the speed of light in vacuum and ϵ_r is the relative permittivity of the dielectric material of the transmission line. If the wire is long enough relative to the signal transition times, a second signal can be driven onto the line before the first signal has been received at the source. The maximum number of bits on the wire is a function of the frequency of operation, wire length, and signal velocity. These factors are related by the following expression:

$$n = \frac{fl}{v} \tag{7.13}$$

where f is the clock frequency, l is the length of the line, and v is the signal velocity.

Given the number of dimensions and layout of the network, the maximum number of bits that can be placed on the wire can be computed from this expression. Typical values for v have been reported as 0.1 – 0.2 m/s [16, 31]. These values provide maximum wire lengths for nonpipelined channels for a 1 GHz switching speed. For $v = 0.2$ m/s and a switching speed of 1 GHz, a wire of length one meter can support the concurrent transmission of 5 bits. While link pipelining has been use in local area networks, until recently its use in multiprocessor interconnects has been quite limited. This is rapidly changing with the continuing increase in clock speeds and confluence of traditional local area interconnect and multiprocessor backplane interconnect technologies [30, 156].

While latency remains unaffected, link throughput would now be decoupled from wire delay and limited by the switching speeds of the drivers and wire media rather than physical length of the wires. Multiple bits can concurrently be in transmission along a wire. From the latency equations developed in earlier sections, it is clear that wire delays play an important role, and as a result, link pipelining fundamentally changes the trade-offs that dictates the optimal number of dimensions. Channel bandwidth is now limited by switching speed rather than wire lengths. Scott and Goodman [310] performed a detailed modeling and analysis of the benefits of link pipelining in tightly coupled multiprocessor interconnects. Their analysis utilized a base case of a 3-D unidirectional torus. To remain consistent with the preceding discussion, we apply their analysis techniques to the 2-D case and latency expressions developed above. Thus, we will be able to compare the effects of link pipelining, switch delays, wiring constraints, and pin-out on the optimal choice of network dimension. The preceding latency expression for wormhole-switched networks can be rewritten as

$$t_{wormhole} = n\frac{k}{4}(r + s + w_{avg}) + s\left\lceil\frac{L}{W}\right\rceil \tag{7.14}$$

The above expression is normalized to the delay between adjacent nodes in a 2-D network. The value of w_{avg} is the average wire delay expressed relative to this base time period. Note the latency term due to message length is now dependent only on the switching delay which determines the rate at which bits can be placed on the wire.

While nonpipelined networks have their clock cycle time dictated by the maximum wire length, pipelined networks have the clock cycle time determined by the switching speed of the routers. The distance component of message latency depends on the length of the wires traversed by the message. However, the wire length between adjacent nodes will depend on the dimension being traversed. The average wire length can be computed as follows [310].

To avoid the long wires due to the wraparound channels, node positions are interleaved as shown in Figure 7.2. Recall that the wire length is computed relative to the wire length between two adjacent nodes in a 2-D mesh. Let the wire length between two nonadjacent nodes in Figure 7.2 be denoted by l_2. In this case there are $(k - 2)$ links of length l_2 and two links of length equal to the base wire length in a 2-D layout (see the 1-D interleaved placement in Figure 7.2). If we assume that this base wire length is one unit and that l_2 is twice the base wire length between adjacent nodes, we can write the average wire length in this dimension as

$$\text{Average wire length} = l_2\frac{k-1}{k} = 2\frac{k-1}{k} \tag{7.15}$$

In practice, l_2 may be a bit longer than twice the base wire length. In the following we retain the variable l_2 in the expressions. Consider topologies with an even number of dimensions. When an n-dimensional topology is mapped into the plane, $\frac{n}{2}$ dimensions are mapped into each physical dimension. As each pair of dimensions are added, the number of nodes in each physical dimension is increased by a factor of k and the average wire length in each dimension grows by a factor of k. Mapping each additional pair of dimensions increases this average wire length in each physical dimension by a factor of k. Consider all the logical dimensions mapped to one of the two physical dimensions. The sum of the mean wire lengths in each of these dimensions is given by

$$l_2\frac{k-1}{k} + l_2\frac{k-1}{k}k + l_2\frac{k-1}{k}k^2 + \ldots + l_2\frac{k-1}{k}k^{\frac{n-2}{2}} \tag{7.16}$$

We can similarly compute the sum of the mean wire length in each dimension mapped to the other physical dimension. The sum of these two expressions divided by the total number of dimensions gives the average wire length of the topology. This simplified expression appears as

$$l_2\frac{k-1}{k}(1 + k + \ldots + k^{\frac{n-2}{2}})\frac{2}{n} \tag{7.17}$$

Using the sum of the geometric series and simplifying, we have

$$w_{avg} = \frac{2l_2(N^{\frac{1}{2}} - 1)}{nk} = \frac{4(N^{\frac{1}{2}} - 1)}{nk} \tag{7.18}$$

Substituting in the equation for latency, we have an expression for the latency in a link-pipelined network that is embedded in two dimensions.

Optimal Number of Dimensions — Summary

To generate an intuition about the effect of physical constraints on network performance we can make the following observations. Message latency can viewed as the sum of two components: a distance component and a message size component represented as

$$\text{Message latency } = \alpha \times Dist(n) + \beta \times Msg\_Length(n) \tag{7.19}$$

For a fixed number of nodes, the function $Dist(n)$ decreases as the network dimension increases. Under bisection width and constant pin-out constraints the function $Msg\_Length(n)$ increases with increasing network dimension since channel widths decrease. Minimum latency is achieved when the components are equal. The coefficient of the distance component is a function of switching, routing, and wire delays: α determines how fast the distance component of latency reduces with increasing dimension. This appeals to intuition as large switching delays encourage the use of a smaller number of intermediate routers by a message, i.e., a larger number of dimensions. Similarly, β, which is a function of switching and wire delay, determines how fast the message component increases with increasing network dimension. The optimal number of dimensions is determined as a result of specific values of router architecture and implementation technology that fix the values of α and β.

It is important to note that these models provide the optimal number of dimensions in the absence of contention. This analysis must be coupled with the detailed performance analysis usually via simulation as described in Chapter 9, or analytic models of contention(e.g., as in [1, 3, 128]).

7.1.6 Packaging Constraints

It is clear that physical constraints have a substantial impact on the performance of the interconnection network. In practice, electronic packaging is a major determinant of these physical constraints. By *packaging* we refer to the hierarchy of interconnections between computational elements. Systems start with communicating elements sharing a silicon surface. Bare die may then be packaged in single-chip carrier or a multichip module [302], which in turn may be mounted on one or both sides of a printed circuit board. Multiple boards are mounted in a card cage, and multiple card cages may be connected and so on. This is graphically illustrated in Figure 7.5. The materials and fabrication technology at each level of this hierarchy are distinct, leading to interconnects with very different physical characteristics and constraints. For example, wire pitch may vary from a few micrometers on a die to a few hundred micrometers on a printed circuit board. Such parameters clearly determine available wiring area and therefore achievable bisection bandwidth. With multichip modules and area array I/Os the number of I/Os available out of a die becomes proportional to chip area rather than chip perimeter [301]. As the preceding analysis has demonstrated, the choice of topology is strongly influenced by such physical constraints and therefore by the packaging technology.

Packaging technology continues to evolve, keeping pace with rapid advances in semiconductor technology. The advent of new packaging technologies, e.g., low-cost multichip modules (MCM), 3-D chip stacks, etc. will lead to major changes in the relationship between these physical constraints, and as a result significantly impact the cost, performance, and reliability of the next generation of systems in general, and multiprocessor interconnects in particular [84, 123]. An accurate analysis of the effect of packaging technology and options will enable the identification of topologies that effectively utilize available packaging technology, where "form fits function [71]."

From the point of view of the network topology, the fundamental packaging question concerns the placement of die, package, board, and subsystem boundaries and the subsequent performance impact of these choices. Die boundaries may be determined in part by technology as we see advances

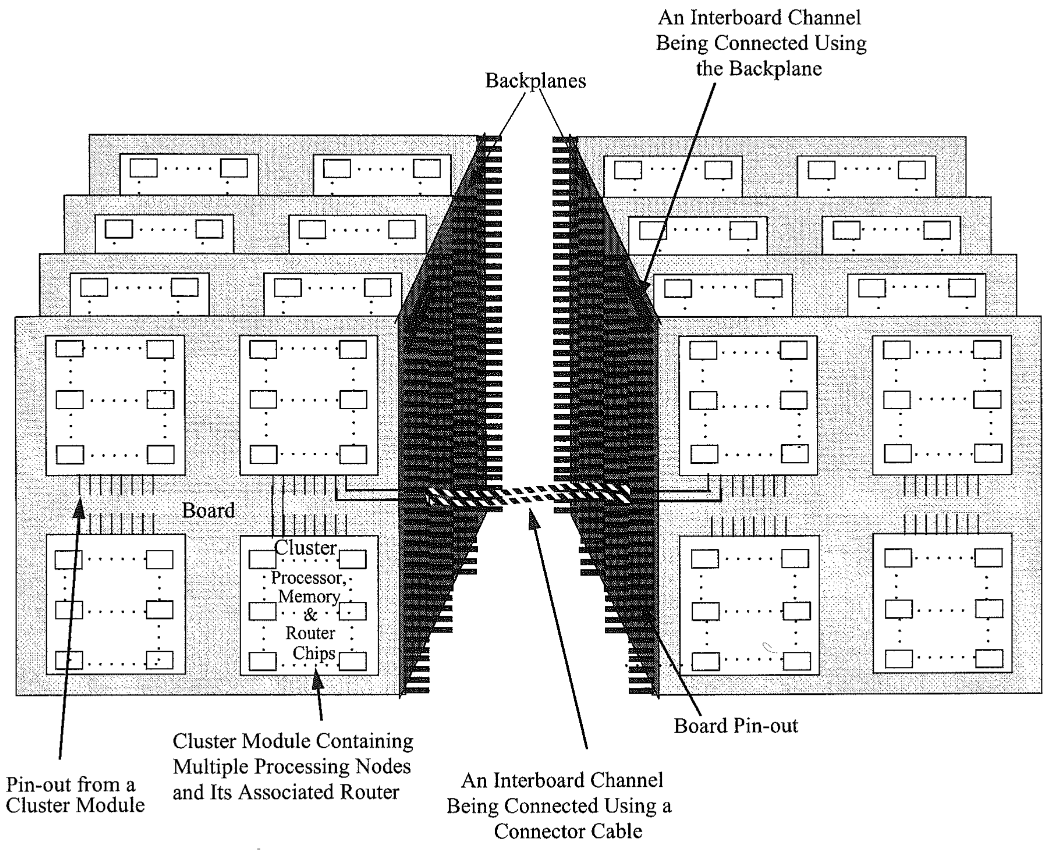

An Interboard Channel Being Connected Using the Backplane
Backplanes
Board
Cluster
Processor, Memory & Router Chips
Board Pin-out
Pin-out from a Cluster Module
Cluster Module Containing Multiple Processing Nodes and Its Associated Router
An Interboard Channel Being Connected Using a Connector Cable

Figure 7.5. Illustration of a packaging hierarchy. (From [19].)

that place multiple processors and the associated routers and interconnect on a single chip. The placement of other boundaries such as multichip carriers and boards are determined by trade-offs in cost, performance, thermal density, reliability, pin-out, etc. For example, the network used to connect 16 processors on a single chip may be quite different from that used on a printed circuit board to connect 64 such chips. Therefore these partitioning decisions are critical. To make the most effective use of packaging technology, we need accurate models of performance to predict the impact of a specific choice of partitions for a given network topology. In a more general setting, such models have been used as elements of a "design for packageability" style of system-level design [84]

Technology projections from groups such as the Semiconductor Industries Association provide assessments of the evolution of implementation technology parameters such as feature size, wafer sizes, clock rates, etc. There have been several recent efforts aimed at assessing the impact of packaging technology on the design and optimization of future network topologies. As a representative example of a relatively new arena of work, the following discussion presents one such approach to assessing the impact of packaging technology on network design.

Basak and Panda [18, 19] provide a model and framework for designing clustered multiprocessor systems in a two-level packaging hierarchy. A k-ary n-cube cluster-c topology is proposed, where each cluster consists of c nodes. While the intracluster network is left undefined and open, the intercluster network is a k-ary n-cube with k^n clusters for a total of $N = k^n \times c$ nodes. With a fixed number of nodes, by varying the number of clusters one can change the physical characteristics of the intercluster and intracluster networks. For example, with single-chip nodes the intracluster network may occupy the surface a board while the intercluster network may appear between boards. In the authors' terminology, the state of the art in packaging at each level offers or supplies physical resources such as bisection bandwidth and pin-out capability. A specific clustering of nodes produces a demand for bisection width, area, and pin-out at each level, e.g., board or chip. The proposed design paradigm is one of reconciling the demand and supply in the most efficient manner possible. Towards this end, analytic models are developed for supplied and demanded resources such as bisection bandwidth, node pin out, and area. Optimization algorithms can utilize these models to produce the most efficient designs.

For example, the demanded bisection bandwidth of an N node system is estimated as follows. Assume that each node injects λ bits/cycle into the network, where a cycle is the wire delay between adjacent routers. As in preceding sections this cycle time is assumed to be normalized to that of the delay between adjacent nodes in a 2-D mesh. On the average, we have $c\lambda$ bits generated within a cluster. With random destinations, a $(1 - \frac{c}{N})$ fraction of these injected bits traverse the intercluster network, and half of this traffic crosses the bisection. Aggregating over all nodes, the demanded bisection bandwidth of a k-ary n-cube cluster-c topology in bits/cycle is

$$\text{Demanded bisection bandwidth} = \frac{N\lambda(1 - \frac{c}{N})}{2} \tag{7.20}$$

As described in [19], the value of λ can be derived from various projections of computation to communication ratios, processor speeds, and emerging network link speeds. Typical estimates can be found in [298]. Now consider the available bisection bandwidth of a packaged clustered network. From the expressions in Table 7.1, we have the bisection bandwidth of the intercluster network as

$$\text{Bisection bandwidth} = \frac{2\frac{N}{c}W}{k_{max}} \tag{7.21}$$

In a regular network where $k_1 = k_2 = \ldots = k_n, k_{max} = (\frac{N}{c})^{\frac{1}{n}}$. The issue here is that the channel width is limited by the pin-out at the chip or board level. In the following, the constraints on the channel width are derived based on a regular network where $k_1 = k_2 = \ldots = k_n$. The extension to distinct radices is straightforward [19]. Consider boards with a capacity of $B = b^n$ clusters, where $b < k$, i.e., no dimension can completely fit on a board and traversal of any dimension forces a interboard communication. Along each dimension we have b^{n-1} nodes with channels traversing interboard boundaries (since the cross section in this dimension has b^{n-1} nodes). Including channels in both directions, the total number of signals coming off the board is given by

$$\text{Board pinout requirements} = 2\sum_{i=1}^{n} b^{n-1}W = 2nb^{n-1}W \tag{7.22}$$

By equating the above to the number of available board-level pins, we can derive the constraint on channel width. The number of available pins may be a function of the perimeter of the board as found in conventional technology, or it may be proportional to the area of the board representing the use of free-space optical communication or the use of elastomeric connectors [255]. The above derivation is based on the assumption that the intraboard bisection width is not the limiting factor

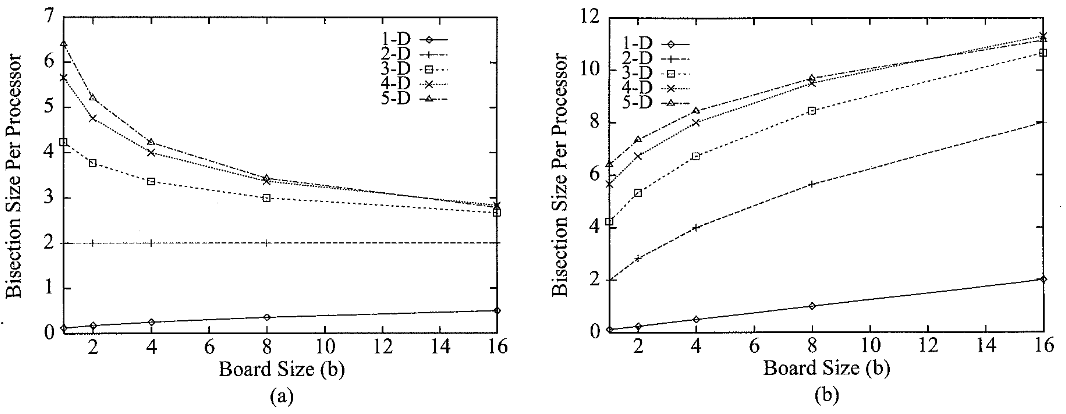


Figure 7.6. Examples of the effect of packaging technology: (a) Periphery pin-out technology. (b) Surface pin-out technology. (From [19].)

on channel width, but rather the interboard pin-out is the limiting factor. This appears to be reasonable in 1997 technology. The alternative is easily modeled by using the intraboard bisection bandwidth as the limiting factor for channel width. Assuming a maximum available board-level pin-out of P_B, we have

$$W = \frac{P_B}{2nb^{n-1}} = \frac{P_B}{2nBb^{-1}} \tag{7.23}$$

Note that B, the number of clusters on a board is a function of the of the board size and cluster size, and therefore a function of packaging technology. By substituting the above expression for channel width in the expression for bisection width, the dependency between bisection width, board size, cluster size, and pin-out can be studied. These expressions relate parameters that define a generation of packaging technology with architectural features that define the network in manner that permits the analysis of the impact of packaging constraints.

An example of the trade-offs that can be studied is illustrated in Figure 7.6. Based on the preceding expressions, these curves from [19] illustrate the relationship between the board size and offered intercluster bisection bandwidth on a per processor basis. This relationship is shown for the cases where the number of I/Os is proportional to board periphery and board area. For the former case, the analysis indicates that for higher-dimensional networks, smaller board sizes offer a higher per-processor bisection bandwidth. The converse is true for the case where the number of I/Os is proportional to board area. A detailed study of the relationships between network and packaging parameters, as well as a precise definitions of the packaging parameters captured in these models can be found in [19].

7.2 Router Architectures

The router architecture is largely determined by the switching technique that is supported. The majority of modern commercial routers found in high performance multiprocessor architectures and emerging switched networks for workstation clusters utilize some form of cut-through switching

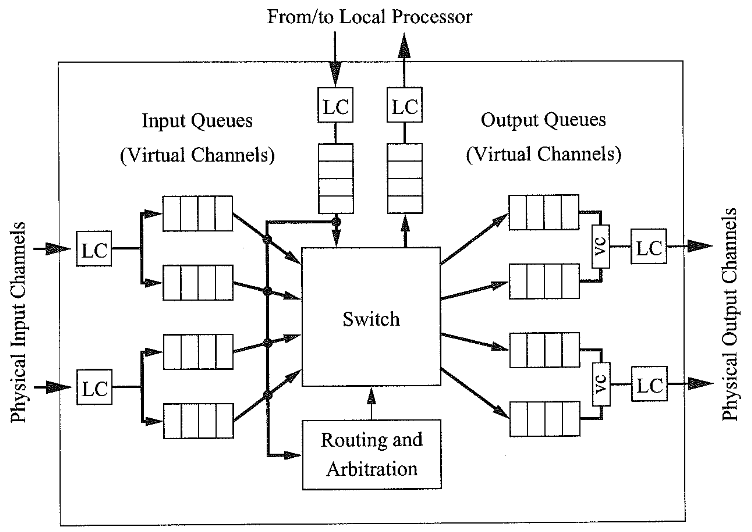


Figure 7.7. Canonical router architecture.

or some variant of it, e.g., VCT switching, wormhole switching, buffered wormhole switching, etc. Therefore this chapter largely focuses on a discussion of issues and designs of such routers. While the router implementations for wormhole and VCT switching differ in many of the architectural trade-offs, they share many common features derived from the use of some form of cut-through switching. The common issues and features facing the design of routers can be categorized as intrarouter or interrouter, and are discussed below. This discussion is followed by descriptions of recent router designs, emphasizing their unique features and reinforcing the commonly held trade-offs.

7.2.1 Intrarouter Performance

In an effort to capture the commonalities and enable quantifiable comparisons, Chien [57] developed an abstract model for the architecture of routers in wormhole-switched k-ary n-cubes. This model is largely concentrated on the intrarouter architecture while the performance of interrouter link operation is very sensitive to packaging implementations. The basic wormhole router functions can be captured in an abstract router architecture as shown in Figure 7.7. We are interested in the implementation complexity of each of the components in the figure.

- *Crossbar switch.* This component is responsible for connecting router input buffers to router output buffers. High-speed routers will utilize crossbar networks with full connectivity, while lower-speed implementations may utilize networks that do not provide full connectivity between input buffers and output buffers.

- *Link controller (LC).* Flow control across the physical channel between adjacent routers is implemented by this unit. The link controllers on either side of a channel coordinate to transfer flow control units. Sufficient buffering must be provided on the receiving side to account for delays in propagation of data and flow control signals. When a flow control event signaling a full buffer is transmitted to the sending controller, there must still be sufficient buffering at the receiver to store all of the phits in transit, as well as all of the phits that will be injected during the time it takes for the flow control signal to propagate back to the sender. If virtual channels are present, the controller is also responsible for decoding the destination channel of the received phit.

- *Virtual channel controller (VC).* This component is responsible for multiplexing the contents of the virtual channels onto the physical channel. With tree-based arbitration, the delay can be expected to be logarithmic in the number of channels.

- *Routing and arbitration unit.* This logic implements the *routing function.* For adaptive routing protocols, the message headers are processed to compute the set of candidate output channels, and generate requests for these channels. If relative addressing is being used in the network, the new headers, one for each candidate output channel, must be generated. For oblivious routing protocols header update is a very simple operation. Alternatively, if absolute addressing is used, header processing is reduced since new headers do not need to be generated.

  This unit also implements the *selection function* component of the routing algorithm: selecting the output link for an incoming message. Output channel status is combined with input channel requests. Conflicts for the same output must be arbitrated (in logarithmic time), and if relative addressing is used, a header must be selected. If the requested buffer(s) is (are) busy, the incoming message remains in the input buffer until a requested output becomes free. The figure shows a full crossbar that connects all input virtual channels to all output virtual channels. Alternatively, the router may use a design where full connectivity is only provided between physical channels and virtual channels arbitrate for crossbar input ports. Fast arbitration policies are crucial to maintaining a low flow control latency through the switch.

- *Buffers.* These are FIFO buffers for storing messages in transit. In the above model, a buffer is associated with both the input physical channels and output physical channels. The buffer size is an integral number of flow control units. In alternative designs, buffers may be associated only with inputs (input buffering) or outputs (output buffering). In VCT switching sufficient buffer space is available for a complete message packet. For a fixed buffer size, insertion and removal from the buffer is usually not on the router critical path.

- *Processor interface.* This component simply implements a physical channel interface to the processor rather than to an adjacent router. It consists of one or more injection channels from the processor and one or more ejection channels to the processor. Ejection channels are also referred to as delivery channels or consumption channels.

While the above router model is representative of routers constructed to date, router architectures are evolving with different implementations. The routing and arbitration unit may be replicated to reduce arbitration time and channels may share ports on the crossbar. The basic functions appear largely unaltered but with distinct implementations to match the current generation of technology.

If the routers support adaptive routing, the presence of multiple choices makes the routing decision more complex. There is a need to generate information corresponding to these choices and to select among these choices. This naturally incurs some overhead in time as well as resources, e.g., chip area. Similarly, the use of virtual channels, while reducing header blocking delays in the network, makes the link controllers more complex by requiring arbitration and more complex flow control mechanisms.

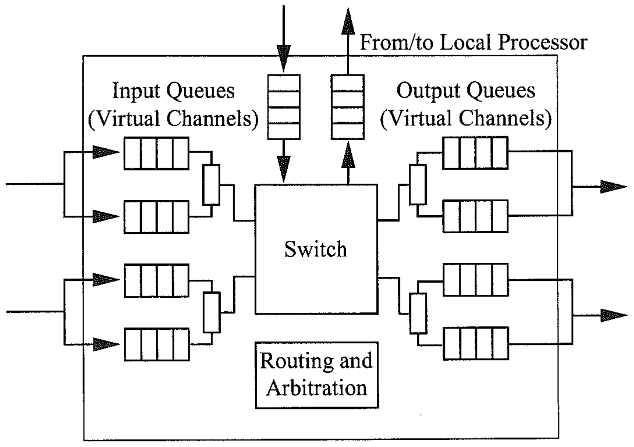
Table 7.4. A parameterization of the component delays in a wormhole-switched router. (From [57].)

Module	Parameter	Gate Count	Delay
Crossbar	P (ports)	$O(P^2)$	$c_0 + c_1 \times \log P$
Flow control unit	None	O(1)	c_2
Address decoder	None	O(1)	c_3
Routing decision	F (freedom)	$O(F^2)$	$c_4 + c_5 \times \log F$
Header selection	F (freedom)	$O(\log F)$	$c_6 + c_7 \times \log F$
VC controllers	V (#VCs)	$O(V)$	$c_8 + c_9 \times \log V$

The two main measures of intrarouter performance [57] are the routing latency or header latency, and the flow control latency. From Figure 7.7 it is apparent that the latency experienced by the header flit(s) through a router is comprised of several components. After the header has been received, it must be decoded and the connection request generated. Since multiple headers may arrive simultaneously, the routing and arbitration unit arbitrates among multiple requests and one of the headers is selected, and routed. This involves computing an output and if it is available, setting the crossbar. The updated header and subsequent data may now be driven through the switch, and will experience some multiplexing delay through the link controllers as they multiplex multiple virtual channels across the physical channel. Once a path has been set up, data flits may now flow through the router with no further involvement with the routing and arbitration unit, but may compete for physical channel bandwidth with other virtual channels.

The flow control latency determines the rate at which flits can be transmitted along the path. In the terminology of pipelines, the flow control latency is the message pipeline stage time. From the figure, we can see that the flow control latency experienced by a data flit is the sum of the delay through the channel flow control, the time to drive the flit through the crossbar, and the multiplexing delay experienced through the link controller, as well as the time to read and write the FIFO buffers. The delay through the link controller is often referred to as the *flit multiplexing delay* or *channel multiplexing delay*. A general model for these delays is quite difficult to derive since the latencies are sensitive to the implementation. One approach to developing such model for wormhole-switched networks is described in [57]. A canonical model of a wormhole-switched router is developed, containing units for address decoding, flow control, and switching. Implementations of the various components (e.g., tree of gates, or selectors for each output) are modeled and expressions for the implementation complexity of each component are derived, but parameterized by technology dependent constants. The parameterized model is illustrated in Table 7.4. By instantiating the constants with values based on a particular implementation technology, realistic delays can be computed for comparing alternative designs. In [57], the values are instantiated for a class of implementations based on gate arrays to derive estimates of flow control latency through the router. The utility of the model stems from the application to many classes of routers. Other router architectures using similar components for partially adaptive or fully adaptive routing can be constructed, and the parameterized model used to estimate and compare intrarouter latencies. In fact, comparisons are possible across technology implementations.

A major determinant of the intrarouter delays is the size and structure of the switch. The use of a crossbar switch to connect all input virtual channels to all output virtual channels is feasible for low-dimensional networks with a small number of virtual channels. A 2-D network with four


From/to Local Processor
Input Queues (Virtual Channels)
Output Queues (Virtual Channels)
Switch
Routing and Arbitration

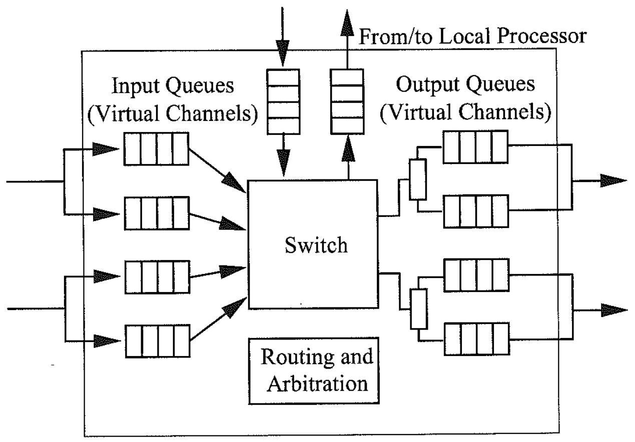

From/to Local Processor
Input Queues (Virtual Channels)
Output Queues (Virtual Channels)
Switch
Routing and Arbitration

Figure 7.8. Alternative router switch organizations.

virtual channels per link would require a 16×16 crossbar and a 3-D network would require a 24×24 switch. Considering that these channels may be byte or word wide, it is apparent that alternatives begin to become attractive. One alternative is to have the switch size determined by the number of physical channels. The data rate into and out of the switch is limited by the bandwidth of the physical channels, while virtual channels decouple buffer resources (and therefore messages) from the physical channel. Thus, channel utilization can remain high, while the construction of crossbar switches remains feasible for higher-dimensional networks. However, input messages must now arbitrate for both switch inputs and outputs. In adaptively routed networks, this arbitration cost grows quadratically with the degree of adaptivity. An intermediate organization provides switch inputs for each input virtual channel, but outputs only for each physical channel. Thus, arbitration is simplified by only having inputs arbitrate for ports on one side of the switch. These alternatives are illustrated in Figure 7.8.

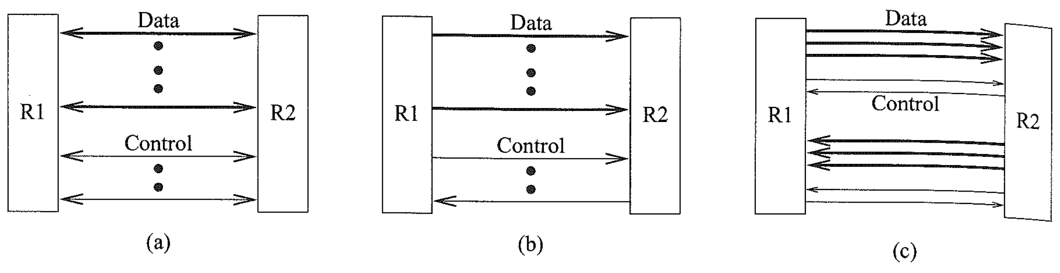


Figure 7.9. An example of: (a) Half-duplex organization. (b) Unidirectional organization. (c) Full-duplex organization.

7.2.2 Physical Channel Issues

The behavioral aspects of interrouter flow control, particularly in the presence of virtual channels is discussed at some length by Dally [73] and Bolding [31] as well as in detail in the descriptions of the individual routers. The following discussion of the issues in physical channel design is largely drawn from these efforts.

In Chapter 2 we have seen that flow control is performed at two levels. Message flow control manipulates logical units of information such as flits. These logical units are the granularity of information for sharing network resources such as interrouter links, intrarouter data paths, and allocating buffers. For example, when a virtual channel competes for, and acquires a physical link, a flit is transferred in its entirety before the control of the channel is relinquished. The transfer of a single flit is not interleaved with other flits. The granularity of a flit determines the effectiveness with which messages share buffers, links, and datapaths. The considerations and performance implications are similar to those facing the choice of thread granularity for multithreaded programs sharing a CPU.

Physical channel flow control may require several cycles to transfer a flit across the channel, e.g., a 64-bit flit across a 16-bit-wide physical channel. In this case, the physical flow control operates on 16-bit phits. By making the flit and phit sizes equivalent, simpler protocols can be used for operating the physical channel. The synchronization of physical transfers can also be used to signify the transfer of flits and the availability of buffer space. Otherwise the message flow control across the channel for blocking messages and allocating and freeing buffers must operate at a different rate than the physical channel signaling rate, increasing the complexity of the router interfaces and intrarouter data paths. In packet-switched and VCT-switched networks, buffer allocation is clearly at the level of logical units such as packets, while channel flow control is typically at finer granularities. In circuit-switched networks, the issue of flow control between routers is encountered during path setup. Once the hardware path has been established message flow control operations are performed between the source and destination routers.

The fixed number of pins devoted to communication between adjacent routers may be organized in several ways. The channel may be organized as (1) a bidirectional half-duplex channel, (2) a unidirectional channel, or (3) a bidirectional full-duplex channel. An example of the three alternatives is illustrated in Figure 7.9. The use of bidirectional half-duplex or unidirectional channels maximizes channel widths. With the exception of pins devoted to control, message transmission between adjacent routers can make full use of the available pins between two routers. However, with unidirectional channels, the topology must be a tori or similar "wrapped" topology to ensure that all

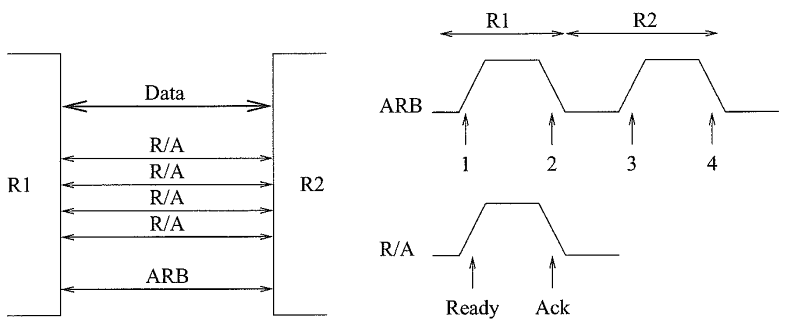


Figure 7.10. An example of the operation of a half-duplex channel.

pairs of nodes may communicate. As a result, the use of unidirectional channels doubles the average distance traveled by a message in tori, increasing the distance component of message latency. Half-duplex links have the disadvantage that both sides of the link must arbitrate for the use of the link. The arbitration must be fair and provide access to both sides of the channel when data are available to be transmitted. Fair arbitration requires status information to be transmitted across the channel to indicate the availability of data to be transmitted. Such traffic may consume bandwidth through messages (in-band transmission) or may be signaled using dedicated pins (out-of-band transmission). We would like this arbitration to be efficient in that the link bandwidth devoted to arbitration signals is minimized. In addition to consuming bandwidth, this arbitration time can increase the flow control latency across the link reducing link utilization as traffic increases. In contrast, this arbitration overhead is avoided in unidirectional full-duplex links, and therefore links can generally be run faster. However, channel widths are reduced i.e., approximately halved as bandwidth is statically allocated in each direction. When the data are being transmitted in only one direction across the channel, 50% of the pin bandwidth is unused. A full-duplex link is only fully utilized when messages are being transmitted in both directions. Depending on the exact pin counts and channel protocol, this organization also serves to roughly double message length and the corresponding component of message latency.

If virtual channels are used, additional bits/signals must be used to distinguish between virtual channels that share the physical channel. These additional bits may use bandwidth or be transmitted across dedicated control pins. An example of the operation of a physical channel that supports multiple virtual channels is provided in Example 7.1.

Example 7.1

The Network Design Frame [79] utilizes half-duplex physical channels to support two virtual channels. The sequence of operations involved in a transfer across this channel are illustrated in Figure 7.10. The channel consists of an 11-bit bidirectional data channel. At power up, one side of the channel is the default owner and has the privilege to transmit information. When the idle side of the channel wishes to transmit a flit, a request is generated by driving the ARB signal high. Channel ownership is transferred when the current owner drives the signal low, and roles are reversed. For example, in the figure $R1$ owns the channel at time 1 and ownership is transferred to $R2$ at time 2. Now $R1$ drives the ARB signal high at time 3 to request ownership and becomes the owner again at time 4. A flit transfer is synchronized

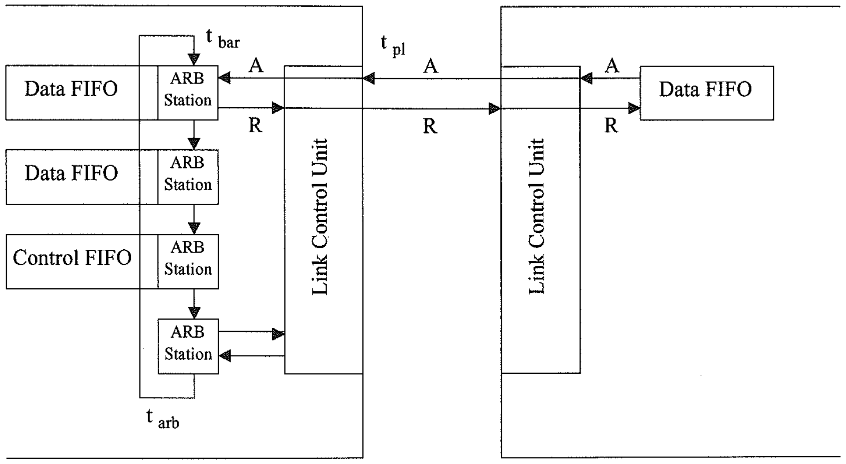


Figure 7.11. An example of overlapping channel arbitration and channel transmission.

using a single request/acknowledge (R/A) signal for each virtual channel in each direction across the physical channel, leading to a total of four R/A signals. A transfer is initiated by driving the corresponding R/A signal high. When the receiving side of the channel is ready to accept another flit, the receiver pulls the R/A signal low.

In the Network Design Frame, one virtual channel has a higher priority than the other. Demand-driven, fair allocation of bandwidth across the physical channel to multiple virtual channels on both sides of the link will require fast, fair arbitration schemes. Ideally we would like arbitration delays to be completely overlapped with transmission and therefore completely hidden. The following example provides one solution for hiding arbitration delays in self-timed router architectures.

Example 7.2

The channel structure and operation of the Ariadne router is illustrated in Figure 7.11 [7]. The operation of this channel is also self-timed, and each virtual channel is granted access to the physical channel using a demand-driven mutual exclusion ring [303]. Within a router output port, each virtual channel owns an arbitration station on the mutual exclusion ring. In addition there is a station on the ring corresponding to the port on the other end of the channel. A signal representing the privilege to drive the physical channel circulates around this ring. The cycle time across a physical link is determined by the interaction between two activities. The first is the arbitration among virtual channels competing for the physical channel. The second is the handshake cycle time for transmission of a flit across the physical channel. A barrier synchronization between these activities can be implemented as part of the channel flow control mechanism as illustrated in Figure 7.11.

The efficiency of this approach can be captured in the following expression. Let t_{arb} represent the period of the arbitration oscillator, i.e., the time for the privilege to circulate through all stations on one side of the channel. Let t_{pl} represents the handshake time to transfer a phit over the physical link and place a request to send the next phit. When a virtual channel is granted the privilege from its ARB Station, phit transmission is stalled until the previous transmission has been completed. When transmission begins, the privilege can be released, serving to overlap transmission latency with the next round of arbitration. When more than one channel is contending for the link, arbitration can be completely hidden. When only one virtual channel is transmitting, the privilege may circulate all the way around the ring before the next phit can be transmitted, potentially slowing down transmission. The privilege is available at intervals of t_{arb}, except the first time when it is available after $\frac{3}{4}t_{arb}$. Thus in general, with one virtual channel transmitting, the privilege is available at points in time of $\frac{3}{4}t_{arb} + mt_{arb}$ for some $m > 0$. Once the physical cycle is complete and the privilege has returned, a small barrier synchronization latency t_{bar} is incurred before the privilege is again released. Thus, the link cycle time with one active virtual channel is

$$t_c = \frac{3}{4}t_{arb} + mt_{arb} + t_{bar}$$

for the minimum m such that $\frac{3}{4}t_{arb} + mt_{arb} + t_{bar} > t_{pl}$. The value of $t_c - t_{pl}$ represents the degree to which arbitration cannot be overlapped with flit transmission across the link.

At low loads, half-duplex links are advantageous since they deliver the full bandwidth of the available pins to the messages whereas full-duplex links remain under utilized. However, at high loads the full bandwidth of the link is utilized in both cases, although arbitration traffic in the case the half-duplex links will produce relatively lower utilization. The disparity in performance is greater when link pipelining is employed [31]. When channel ownership must be switched in the case of a half-duplex link, transmission can begin only after the channel has been drained of the bits currently in transit. In addition, the first phit transmitted in the new direction incurs the full latency of the physical link. Thus, arbitration overheads relative to full-duplex operation correspondingly increase since a greater percentage of time is spent in switching directions across the channel. A second problem is in the propagation of status information. With pipelined links, the received status of flow control information may not reflect the actual status at the time it is received. This is an issue for both half-duplex and full-duplex pipelined links.

For example, the transmission of flow control information to indicate the lack of buffer space arrives too late at the sender: multiple phits are already in transit and will be dropped due to lack of space. However, full link utilization can be maintained without loss with a buffer on the receiver of sufficient depth to store all the phits in transit across the channel. The relationship between the buffer depth, wire delay, bandwidth, etc. can be derived as follows [107]. Assume that propagation delay is specified as P ns per unit length and the length of the wire in the channel is L units. When the receiver requests the sender to stop transmitting, there may be LPB phits in transit on a channel running at B Gphits/s. In addition, by the time the flow control signal propagates to the sender, an additional LPB phits may be placed in the channel. Finally, if the processing time at the receiver is F ns (e.g., flow control operation latency), this delay permits the sender to place another FB phits on the channel. The available buffer space in the receiver must be large enough to receive all of these phits when a *stop* signal is transmitted to the sender. These *slack buffers* [314] serve to hide the wire latency and the flow control latency. The size of the buffers can be estimated as follows.

$$\text{Available buffer space} = 2LPB + FB \text{ phits} \tag{7.24}$$

In cases where long wire latencies dominate, the use of link pipelining favors the use of full-duplex channels, particularly under high loads. This is typically the case in the emerging class of low latency interconnects for workstation/PC clusters. To reduce the wiring costs of long interconnects as well

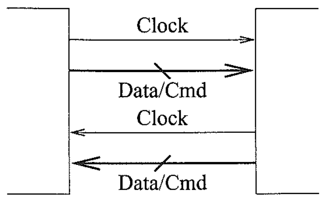


Figure 7.12. An example of the operation of a full-duplex channel with data/command encoding.

as maintain high link utilization, full-duplex channel techniques have been developed using slack buffers and pin-efficient flow control. An example is described in Example 7.3.

Example 7.3

The structure of an example full-duplex channel is shown in Figure 7.12. This channel is used in the routing fabric of Tandem's ServerNet [156], and differs from the preceding example is several ways. Physical channel flow control is synchronous and the clock is transmitted across the link. At the transmitting end of the channel, the clock is sent through similar delay paths as the data so that it may be used to sample the data at the receiver. Buffer management and message-level flow control is via commands that are transmitted in the reverse direction across the 9-bit data/command channel. The 9-bit encoding provides for 256 data symbols and 20 command symbols. The specific encoding used in ServerNet is shown below.

Bit 8	Bit 7	Bit 6	Function
0	0	0	Command
0	0	1	Error
0	1	0	Error
1	0	0	Error
0	1	1	Data <7:6> = 00
1	0	1	Data <7:6> = 01
1	1	0	Data <7:6> = 10
1	1	1	Data <7:6> = 11

The most significant 3 bits determine how the remaining 6 bits will be interpreted. For command words, these 6 bits are encoded as a 3-of-6 code where all command words have exactly three 1s and three 0s, resulting in 20 such command words. Examples of command symbols that may be used include *STOP, SEND, END-OF-MSG*, and *IDLE*. The data symbols are encoded as four groups of 64 symbols. For example, from the above table reception of 011101011 would be interpreted as the data word 00101011, while reception of 110101011 would be interpreted as 10101011. The error codes are used to ensure a minimum Hamming distance of two between command symbols and data symbols. Commands interspersed within the data stream are used for buffer management and message flow control in addition to error signaling. Slack buffers are used to hide the latency of the transmission of flow control commands.

A similar physical channel structure and approach is used within the routers for Myrinet. Both Tandem's ServerNet and Myricom's Myrinet bring traditional multiprocessor network technology to the workstation and PC cluster environment. Interconnects can be substantially longer and therefore the delays for asynchronous handshaking signals become larger. The use of synchronous channels and slack buffers to overlap flow control delays as described above is motivated in part by operation in this high latency environment. Another advantage of encoding commands is the reduction in pin-out requirements. The percentage of pins devoted to data is higher although some of the bandwidth is now devoted to control.

There are also many other technological features that impact the choice of full- or half-duplex links. We point to one technique for maximizing the use of available pins on the router chips: simultaneous bidirectional signaling [48, 76, 193]. This technique allows simultaneous signaling between two routers across a single signal line. Thus, full-duplex bidirectional communication of a single bit between two routers can be realized with one pin (signal) rather than two signals. This is achieved by transmitting [76] a logic 1 (0) as a positive (negative) current. The received signal is the superposition of the two signals transmitted from both sides of the channel. Each transmitter generates a reference signal which is subtracted from the superimposed signal to generate the received signal. The result is a considerable savings over the number of I/O pins required, and consequent reduction in the packaging cost. The low voltage swing employed in the Reliable Router [76] results in additional reduction in power consumption, but necessitates different noise reduction techniques. This approach would enable wider, half-duplex links and larger number of dimensions. Currently the main difficulty with this scheme is the complexity of the signaling mechanism and the susceptibility to noise. Both of these disadvantages can be expected to be mitigated as the technology matures.

Collectively, from the preceding observations we can make the following general observations. For large systems with a higher number of dimensions, wire delays would dominate. The use of higher clock speeds, communication intensive applications and the use of pipelined links in this environment would favor full-duplex links. Lower dimensionality, and communication traffic below network saturation would tend to favor the use of half-duplex links. Cost considerations would encourage the use of low(er) cost packaging which would also favor half-duplex links and the use of command/data encodings to reduce the overall pin count and therefore package cost.

7.2.3 Wormhole Routers

A Generic Dimension-Ordered Router and its Derivatives

A generic dimension-ordered router can be represented as shown in Figure 7.13 [57]. The abstract router design in Figure 7.7 can be refined to take advantage of the structure of dimension-ordered routing. In dimension-ordered routing, messages complete traversal in a dimension prior to changing dimension. Thus, it is not necessary to provide paths from any input link to any output link. An incoming message on dimension X will either continue along dimension X (continuing in either the positive or negative direction), or proceed to the next dimension to be traversed. Thus, each dimension can be serviced by a 3×3 crossbar as shown in Figure 7.13 for a 2-D router. The first dimension accepts messages from the local node and the last dimension ejects messages to the local node. A message must cut through each dimension crossbar. For example, consider a message that is injected into the network that must only traverse the Y dimension. It is injected into the X dimension crossbar where routing logic forwards it to the Y dimension crossbar, and subsequently out along the Y dimension link.

If dimension offsets are used in the header, routing decisions are considerably simplified. A check for zero determines if the traversal within the dimension has been completed. The header

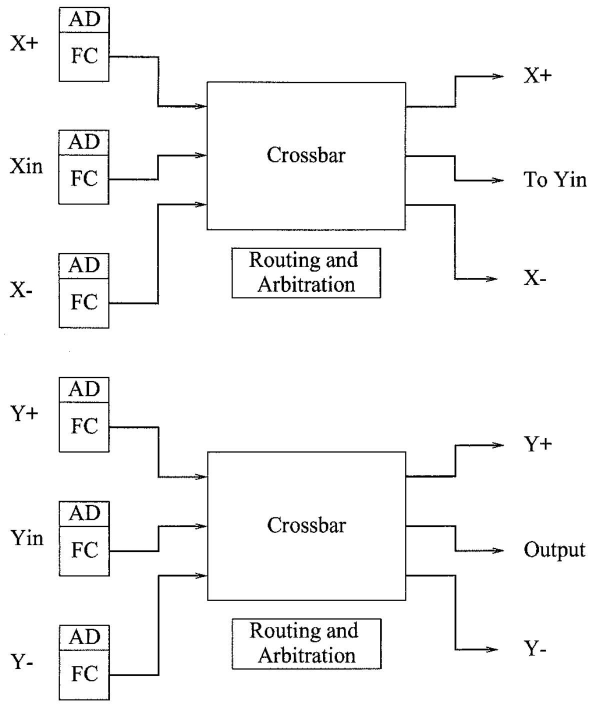


Figure 7.13. Organization of a canonical wormhole router. (AD = Address decoder; FC = Flow control.)

is updated and forwarded through the output port. An example of a router that uses such a partitioned data path is the network design frame [79]. This is a router designed for use in 2-D mesh-connected networks. Each physical channel is an 11-bit bidirectional channel supported by five control lines to implement channel flow control. Each link supports two virtual channels in each direction. Four of the control signals are bidirectional lines used for request/acknowledge flow control for each virtual channel. The fifth bidirectional control line is used to implement a token-passing protocol to control access to the bidirectional data lines. The physical channel flow control protocol is described in Example 7.1. The two virtual channels implement two virtual networks, at two priority levels. Latency-sensitive traffic uses the higher priority network. Since physical bandwidth is rarely completely utilized, the use of virtual channels to separate traffic is an attractive alternative to two distinct physical networks. Within the router, each virtual network implementation is well represented by the canonical organization shown in Figure 7.13. A 3×3 nonblocking switch (reference the multistage topologies in Chapter 1) can be implemented with four 2×2 switches. Using the partitioned design in Figure 7.13, the four crossbar switches can be implemented using twelve 2×2 switching elements, rather than having a 9×9 full crossbar switch with 11-bit channels at each port. This represents an area savings of approximately 95%.

The first two flits of the message header contain the offsets in the X and Y direction respectively. The routing units perform sign and zero checks to determine if the message is to continue in the same dimension, transition to the next dimension, or be ejected. Output link controllers perform

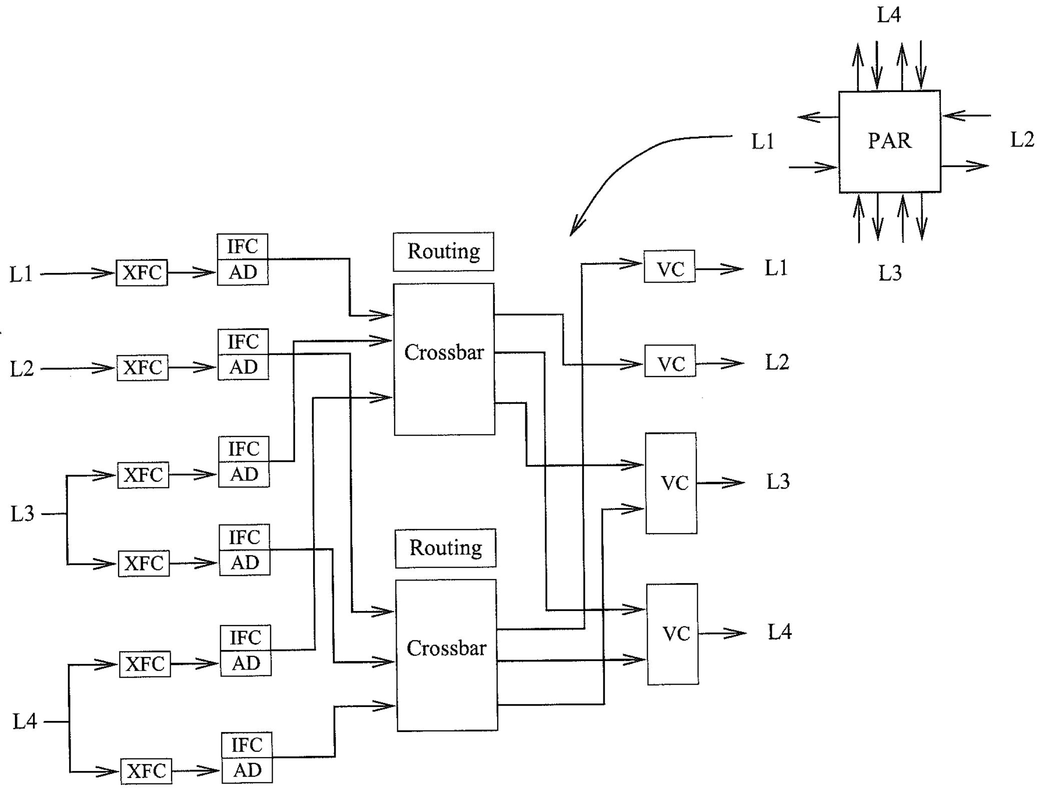


Figure 7.14. Organization of a planar-adaptive router. (AD = Address decoder; IFC = Internal flow control; PAR = Planar-adaptive router; VC = Virtual channel controller; XFC = External flow control.)

arbitration between the two virtual channels to grant access to the physical link. If the router does not have access to the bidirectional channel, there is a delay before the token is requested and transferred from the other side of the channel.

These wormhole routers exploit the routing restrictions of dimension-ordered routing to realize efficient router architectures. If adaptive routing is permitted, the internal router architectures begin to become more complex, especially as the number of dimensions and the number of virtual channels increase. This is simply a direct result of the increased routing flexibility. However, the idea of using partitioned datapaths to reduce the implementation complexity can still be applied if adaptivity can be controlled. The advantages of the partitioned datapath found in dimension-ordered routers can be combined with limited adaptive routing to achieve a more balanced, less complex design. The planar-adaptive routing techniques described in Chapter 4 is an attempt to balance hardware complexity with potential advantages of increased routing freedom. The basic strategy is to adaptively route in a sequence of adaptive planes. The block diagram of a 2-D router is shown in Figure 7.14 [11]. Note how the datapath is partitioned. Each crossbar provides the switching for the increasing or decreasing network. Messages only travel in one or the other. Higher-dimensional networks can be supported by providing the same basic organization in each successive pair of dimensions. This technique can also be generalized to support higher degrees of adaptivity by increasing the number of dimensions within which adaptive routing is permitted. Figure 7.14 illustrates the organization for adaptive routing in two dimensions at a time. Alternatives may permit adaptive routing in three dimensions at a time. A 5-D network may permit adaptive routing in successive 3-D cubes. These routers are referred to as *f-flat* routers, where the parameter f signifies the number of dimensions within which adaptive routing is permitted at any one time. Aoyama and Chien [11] use this baseline design to study the performance impact of adaptive routing via the design of f-flat routers. While performance may be expected to improve with increasing adaptivity, the hardware complexity certainly grows with increasing f.

This idea of using partitioned datapaths to reduce internal router complexity can be used in fully adaptive routers as well in order to offset the competing factors of increased adaptivity and low router delay. The internal router architecture can be designed to fully exploit the expected routing behavior as well as the capabilities of the underlying routing algorithm to achieve highest possible performance. For instance, if most packets tend not to change dimensions or virtual channel classes frequently during routing (i.e., routing locality in dimension or in virtual channel class), then it is not necessary for the internal datapath to provide packets with direct access to all output virtual channels, even in fully adaptive routing. Each crossbar switch may provide access to all the virtual channels in the same dimension. Alternatively, a given crossbar switch may provide access to a single virtual channel in each dimension. In both cases, each switch must provide a connection to the next switch.

For instance, in the Hierarchical Adaptive Router [215], the internal datapath is partitioned into ordered virtual channel networks which includes all dimensions and directions. Deadlock is avoided by enforcing routing restrictions in the lowest virtual network. Although this router architecture exploits routing locality in virtual channel class, the lowest virtual channel network (the escape resource) presents a potential bottleneck. While this ordering between virtual channel networks is required for deadlock avoidance-based routing, it is a restriction that can be relaxed in recovery-based routing. Choi and Pinkston [61] proposed such an enhancement for a Disha deadlock recovery-based router design.

The Intel Teraflops Router

The Teraflops system is an architecture being designed by Intel Corporation in an effort to produce a machine capable of a sustained performance of 10^{12} floating-point operations per second and a peak performance of 1.8×10^{12} floating-point operations per second. *Cavallino* is the name for the network interface chip and router that forms the communication fabric for this machine [48]. The network is a k-ary 3-cube, with a distinct radix in each dimension. Up to eight hops are permitted in the Z dimension, 256 hops in the X dimension and 64 hops in the Y dimension. The router has six ports and the interface design supports 2-D configurations where the two ports in the Z dimension are connected to two processing nodes rather than other routers. This architecture provides a flexible basis for constructing a variety of topologies.

Unlike most other routers, Cavallino uses simultaneous bidirectional signaling. Physical channels are half-duplex. A receiver interprets incoming data with respect to the reference levels received and the data being driven to determine the value being received. The physical channel is 16 bits wide (phits). Message flits are 64 bits and require four clock cycles to traverse the channel. Three additional signals across the channel provide virtual channel and flow control information. The buffer status transmitted back to the sending virtual channel is interpreted as almost full rather than an exact number of available locations. This scheme allows for some slack in the protocols to

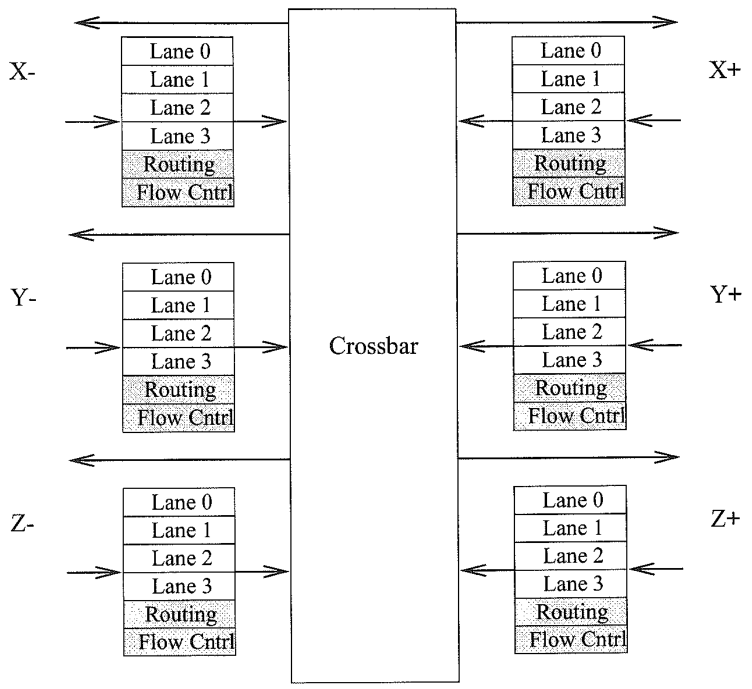


Figure 7.15. Architecture of the Intel teraflops router.

tolerate signaling errors. The physical channel operates at 200 MHz, providing an aggregate data rate of 400 Mbytes/s in each direction across the link. Four virtual channels are multiplexed in each direction across a physical link.

The internal structure of the router is shown in Figure 7.15. A message enters along one virtual channel and is destined for one output virtual channel. The routing header contains offsets in each dimension. In fact, routing order is $Z-X-Y-Z$. Using offsets permits fast routing decisions within each node as a message either turns to a new dimension or continues along the same dimension. The router is input buffered. With six input links and four virtual channels/link, a 24-to-1 arbitration is required on each crossbar output. The speed of operation and the length of the signal paths led to the adoption of a two-stage arbitration scheme. The first stage arbitrates between corresponding virtual channels from the six inputs, e.g., a 6-to-1 arbiter for virtual channel 0 on each link. The second stage is a 4-to-1 arbitration for all channels with valid data competing for the same physical output link. The internal data paths are 16 bits, matched to the output links. The ports were designed such that they could be reset while the network was operational. This would enable removal and replacement of routers and CPU boards while the network was active.

A block diagram of the network interface is shown in Figure 7.16. The router side of the interface utilizes portions of the router architecture configured as a three-port design. One port is used for message injection and ejection while two other ports are used for connection to two routers. This flexibility permits the construction of more complex topologies. This is often desired for the purposes of fault tolerance or simply increased bandwidth. Each output virtual channel is 384 bytes deep while

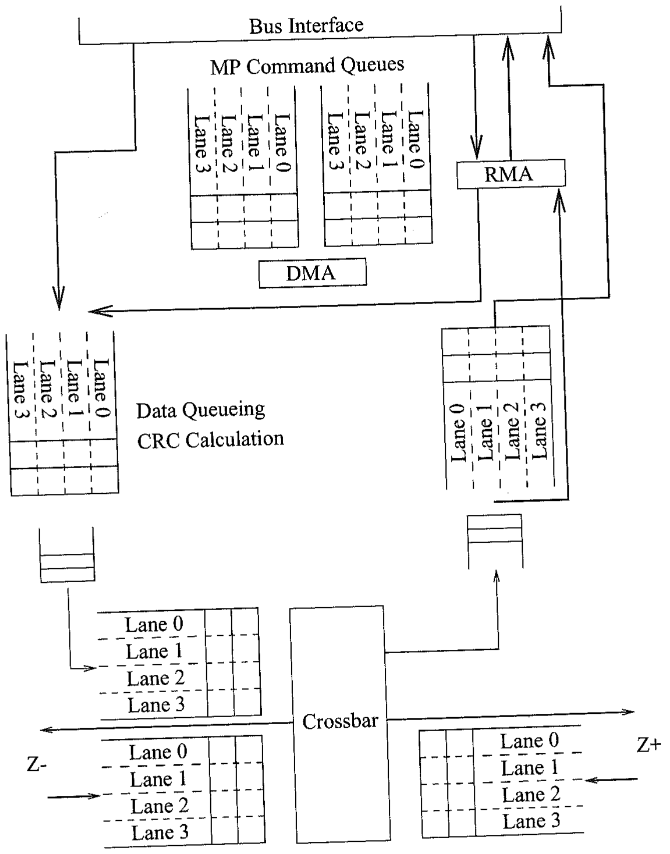


Figure 7.16. Interface architecture for the Cavallino router. (CRC = Cyclic redundancy check.)

each input virtual channel is 256 bytes deep. There are two additional interesting components of the interface. The first is the remote memory access (RMA) engine. The RMA engine has access to a memory-resident mapping table that maps processor addresses into the physical addresses of remote nodes. Request messages are created for remote accesses and response messages are created to service remote requests. The RMA engine also provides support for remotely accessible atomic operations, e.g., read and clear. This functionality supports synchronization operations. The second interesting feature is the direct memory access (DMA) interface. The DMA engine supports eight channels and corresponding command queues (one for each input and output virtual channel). Each channel can store up to eight commands. This volume of posted work is supported by command switching by the DMA engine, e.g., when injection is blocked due to congestion. The access to the control of the DMA engine is memory-mapped, and user access is permitted to support the implementation of low-overhead, lightweight messaging layers. Throughput through the DMA engine approaches 400 Mbytes/s.

While forming the backbone of the Teraflops machine, the Cavallino chip set is also regarded as a high-performance chip set that can be used for configuring high-performance NOWs.

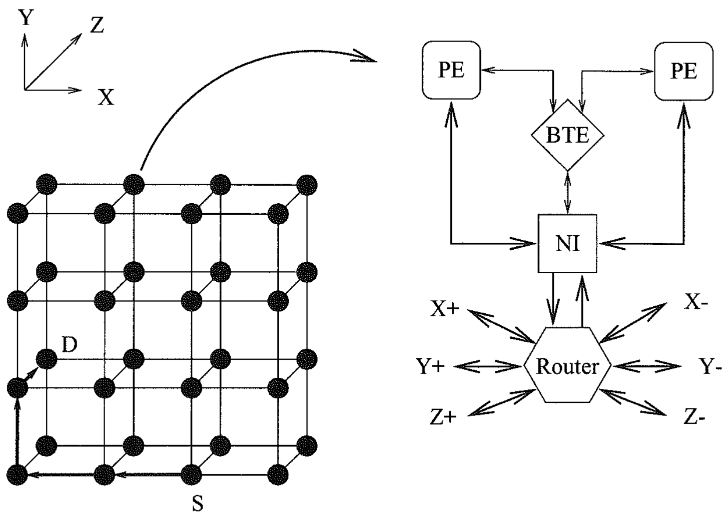


Figure 7.17. Organization of the T3D system. (BTE = Block transfer engine; NI = Network interface.)

The Cray T3D

The Cray T3D utilizes a k-ary 3-cube interconnect. An example of a $4 \times 4 \times 2$ configuration of processing element (PE) nodes is is illustrated in Figure 7.17. Each PE node consists of two DEC Alpha 150 MHz processors sharing a single network interface to the local router. The network has a maximum radix of 8 in the X and Z dimensions and 16 in the Y dimension. Therefore, the maximum configuration is 1,024 PE nodes or 2,048 processors. The system is packaged with two PE nodes per board. This choice of radix is not arbitrary. Rather it is based on the observation that under random traffic, messages will travel $\frac{1}{4}$ the way around each dimension in the presence of bidirectional links. If we consider only the packets that travel in say, the $X+$ direction, then the average distance traveled by these packets will be $\frac{1}{8}$ the distance in that direction. If the injection and ejection ports of the network interface have the same capacity as the individual links then a radix of 8 in each dimension can be expected to produce a balanced demand on the links and ports.

The router architecture is based on fixed-path, dimension-order wormhole switching. The organization of the relevant components is shown in Figure 7.18. There are four unidirectional virtual channels in each direction over a physical link. These four virtual channels are partitioned into two virtual networks with two virtual channels/link. One network is referred to as the request network and transmits *request packets.* The second network is the reply network carrying only *reply packets.* The two virtual channels within each network prevent deadlock due to the wraparound channels as described below. Adjacent routers communicate over a bidirectional, full-duplex physical channel, with each direction comprised of 16 data bits and 8 control bits as shown in Figure 7.18. The 4 forward control channel bits are used to identify the type of message packet (request or response) and the destination virtual channel buffer. The four acknowledge control signals are used for flow control and signify empty buffers on the receiving router. Due to the time required for channel flow control (e.g., generation of acknowledgments), as long as messages are integral multiples of 16 bits (see message formats below), full link utilization can be achieved. Otherwise some idle physical

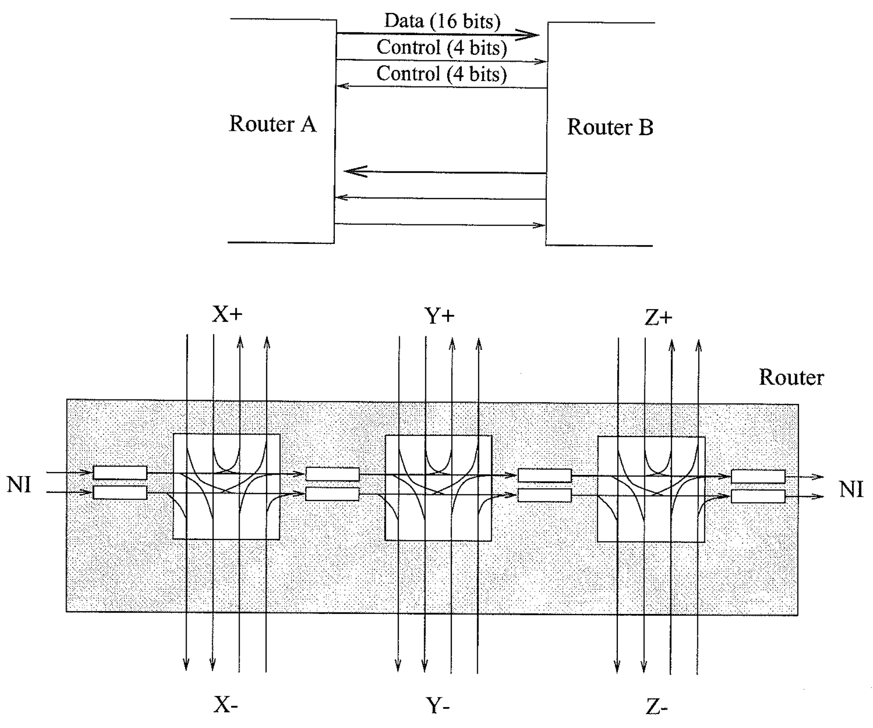


Figure 7.18. Architecture and operation of the T3D router.

channel cycles are possible. The router is physically partitioned into three switches as shown in the figure. Messages are routed to first correct the offset in X, then Y, and finally Z dimensions. The first switch is connected to the injection channel from the network interface and is used to forward the message in the $X+$ or $X-$ directions. When the message has completed traversal in the X dimension, the message moves to the next switch in the intermediate router for forwarding along the Y dimension, and then to the Z dimension. Message transition between switches is via *interdimensional virtual channels.* For the sake of clarity the figure omits some of the virtual channels.

Message packets comprise of a header and body. Sample formats are shown in Figure 7.19. The header is an integral number of 16-bit phits that contain routing information, control information for the remote PE, and possibly source information as well. The body makes use of check bits to detect errors. For example, a 64-bit word will be augmented with 14 check bits (hence the 5-phit body in Figure 7.19) and four-word message bodies will have 56 check bits (necessitating 20-phit bodies). The messages are organized as sequences of flits, with each flit comprised of 8 phits. Virtual channel buffers are 1 flit deep.

The T3D utilizes *source routing.* Header information identifies the sequence of virtual channels that the message will traverse in each dimension. In order to understand the routing strategy, we must first be familiar with the PE addressing scheme. While all PEs have physical addresses, they also possess logical addresses and virtual addresses. Logical addresses are based on the logical

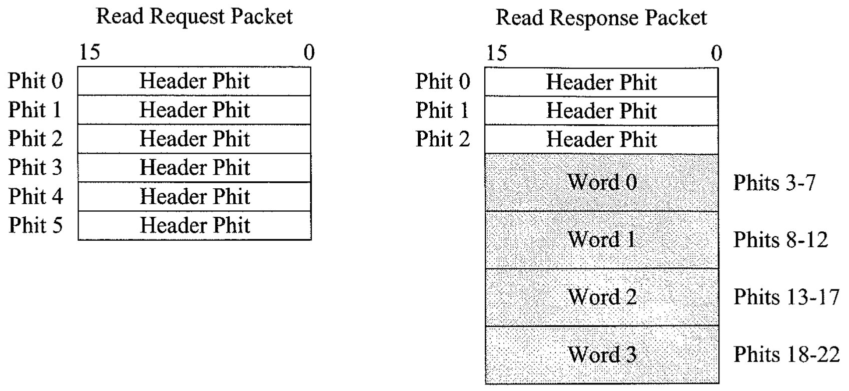


Figure 7.19. T3D packet formats.

topology of the installation. Thus, nodes can be addressed by the coordinate of their logical position in this topology, and are in principle independent of their physical location. This approach permits spare nodes to be mapped into the network to replace failed nodes by assigning the spare node the same logical address. The virtual address is assigned to nodes within a partition allocated to a job. The virtual address is interpreted according to the shape of the allocated partition. For example, 16 nodes can be allocated as an $8 \times 2 \times 1$ topology, or as a $4 \times 2 \times 2$ topology. In the latter case the first 2 bits of the virtual address are allocated to the X offset and the remaining 2 bits to the Y and Z offsets. When a message is transmitted, the operating system or hardware translates the virtual PE address to a logical PE address. This logical PE address is used as an index into a table that provides the offsets in each dimension to forward the message to the correct node. In this manner if a spare PE is mapped in to replace a faulty PE, the routing tables at each node simply need to be updated.

Dimension-order routing prevents cyclic dependencies between dimensions. Deadlock within a dimension is avoided by preventing cyclic channel dependencies as shown in Figure 7.20. One channel is identified by the software as the *dateline* communication link. Consider a request message that

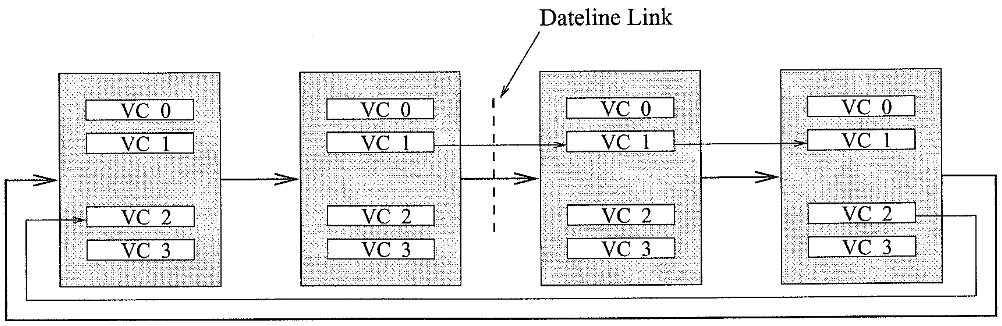


Figure 7.20. Deadlock avoidance within a dimension. (VC = Virtual channel.)

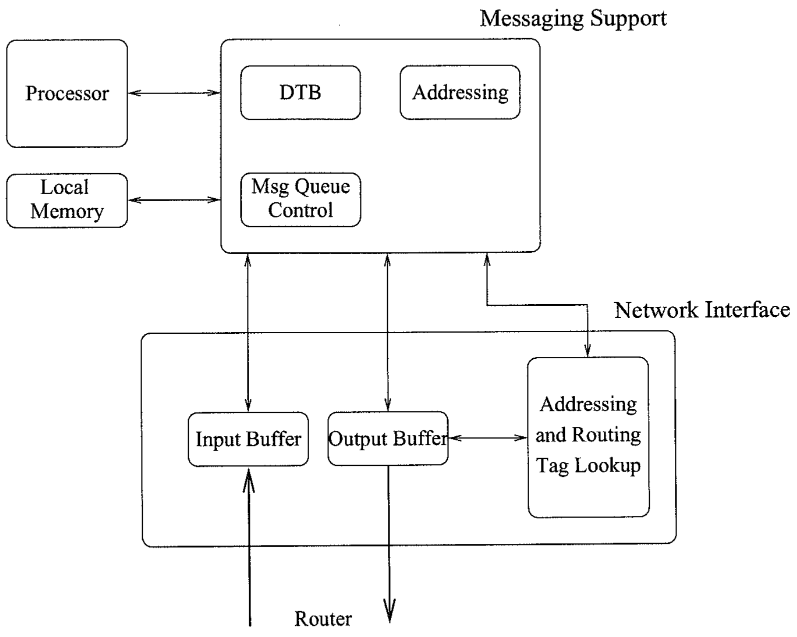


Figure 7.21. Message processing datapath. (DTB = Data translation buffer.)

must traverse this dimension. If the message will traverse the dateline communication link, it will be forwarded along virtual channel 1, otherwise it will be forwarded along virtual channel 0. The message does not switch between virtual channels within a dimension. The dateline effectively identifies the wraparound channel, and the routing restrictions implement the Dally and Seitz [78] restrictions to guarantee deadlock-free routing. When the routing tables are constructed, they implement these restrictions. Such restrictions, while realizing deadlock-free routing, result in unbalanced utilization of the virtual channels. The flexibility of source-based routing enables static optimization of the message paths to balance traffic across virtual channels and further improve link utilization [311].

The T3D provides architectural support for messaging. The routing tables are supported in hardware, as is virtual to logical PE address translation prior to routing tag lookup. Message headers are constructed after the processor provides the data and address information. When a message is received, it is placed in a message queue in a reserved portion of memory and an interrupt is generated. If the store operation to the message queue fails, a negative acknowledgment is returned to the source where messages are buffered to enable retransmission. Successful insertion into the queue generates a positive acknowledgment to the source. Request and reply packets can implement a shared address space (not coherent shared memory). When the processor generates a memory address, support circuitry maps that address into a local or nonlocal address. Nonlocal references result in request packets being generated to remote nodes. A block diagram illustrating the organization of the support for message processing within the node is shown in Figure 7.21.

The T3D links operate at 150 MHz or 300 Mbytes/s. The reported per hop latency is two clock cycles [311] while the maximum sustained data bandwidth into the node (i.e., excluding packet headers, acknowledgments, etc) is 150 Mbytes/s per node or 75 Mbytes/s per processor.

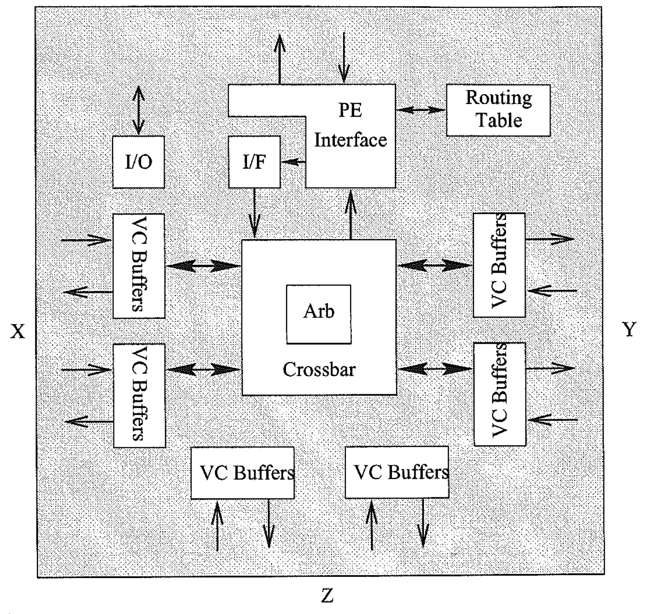


Figure 7.22. Architecture of the T3E router. (Arb = Arbitrator; I/F = Interface.)

The Cray T3E

The T3E represents a second-generation multiprocessor system [309, 312]. The network and router architecture of the T3E retains many of the operational characteristics of the T3D but is very different in other ways. The system design evolved to substantially improve latency tolerance, resulting in a shift in emphasis in the design of the network from reducing latency to providing sustained bandwidth. This resulted a change in the balance between processors and network bandwidth — only one processor/network node in the T3E — and the use of standard-cell CMOS (single-chip) rather than ECL (three-chip) technology. Along with doubling of the link bandwidth, this produces a factor of four increase in the per processor bandwidth. The topology is still that of a 3-D torus with a maximum radix of 8, 32, and 8 in the X, Y, and Z dimensions, respectively. A block diagram of the T3E router is shown in Figure 7.22.

A T3E packet is 1 – 10 flits, and each flit is 5 phits. The network physical channel is full-duplex with 14-bit data producing a flit size of 70 bits. A flit can carry one 64-bit word with some additional control information. The router operates on a 75 MHz clock. During each clock cycle 5 phits are transmitted across the channel, leading to a link transmission rate of 375MHz and link bandwidth of 600 Mbytes/s.

Five virtual channels are multiplexed in each direction over each physical channel. Four of the channels essentially function as they do in the T3D (with differences identified below) while the fifth channel is used for fully adaptive routing. Each adaptive channel can buffer up to 22 flits while each of the regular channels can buffer up to 12 flits. The channels utilize credit based flow control for buffer management with acknowledgments in the reverse direction being piggybacked on other messages or transmitted as idle flits. Round-robin scheduling across active virtual channels within a group determines which channel will compete for crossbar outputs. The winning channel can request a virtual channel in the deterministic direction, or an adaptive channel in the highest-ordered direction yet to be satisfied. If both are available, the adaptive channel is used. Once an output virtual channel is allocated, it must remain allocated, while other virtual channel inputs may still request and use the same physical output channel. Finally, output virtual channels must arbitrate for the physical channel. This process is also round robin although the adaptive channel has the lowest priority to start transmitting.

Routing is fully adaptive. The deterministic paths use four virtual channels for the request/reply network as in the T3D. However, the path is determined by ordering dimensions *and* the direction of traversal within each dimension. This style of routing can be regarded as *direction order* rather than dimension order. The ordering used in the T3E is $X+$, $Y+$, $Z+$, $X-$, $Y-$, and $Z-$. In general, any other ordering would work just as well. By applying the concept of channel classes from Chapter 3, it is apparent that each direction corresponds to a class of channels, and the order in which channel classes are used by a message is acyclic. Thus, routing is deadlock-free. In addition, the network supports an initial hop in any positive direction, and permits one final hop in the $Z-$ direction at the end of the route. These initial and final hops add routing flexibility and are only necessary if a normal direction-order route does not exist. Each message has a fixed path that is determined at the source router. A message can be routed along the statically configured path, or along the adaptive virtual channel over any physical link along a minimal path to the destination. However, the implementation does not require the extended channel dependency graph to be acyclic. The adaptive virtual channels are large enough to buffer two maximal messages. Therefore indirect dependencies between deterministic channels do not exist. This permits optimizations in the manner in which virtual channels are assigned to improve virtual channel utilization and minimize imbalance due to routing restrictions. Finally, adaptive routing can be turned off via control fields within the routing tables, and on an individual packet basis through a single bit recorded in the message. Since adaptively routed messages may arrive out of order at the destination, the latter capability enables the processor to issue sequences of messages that are delivered in order.

Memory management and the network interface operation have been made more flexible than the first-generation support found in the T3D. An arbitrary number of message queues are supported in both user memory and system memory. Message queues can be set up to be interrupt-driven, polled, or interrupt-driven based on some threshold on the number of messages. In general, message passing is more tightly coupled with operations for the support of shared-memory abstractions.

The Reliable Router

The Reliable Router chip is targeted for fault-tolerant operation in 2-D mesh topologies [76]. The block diagram of the Reliable Router is shown in Figure 7.23. There are are six input channels corresponding to the four physical directions in the 2-D mesh, and two additional physical ports: the local processor interface, and a separate port for diagnostics. The input and output channels are connected through a full crossbar switch, although some input/output connections may be prohibited by the routing function.

While message packets can be of arbitrary length, the flit length is 64 bits. There are four flit types: head, data, tail, and token. The format of the head flit is as shown in Figure 7.24. The size of the physical channel or phit size is 23 bits. The channel structure is illustrated in Figure 7.23. To permit the use of chip carriers with fewer than 300 pins, these physical channels utilize half-duplex channels with simultaneous bidirectional signaling. Flits are transferred in one direction across the physical channel as four 23-bit phits called *frames*, producing 92-bit transfers. The format of a data flit and its constituent frames are as shown in Figure 7.25. The 28 bits in excess of the flit size are

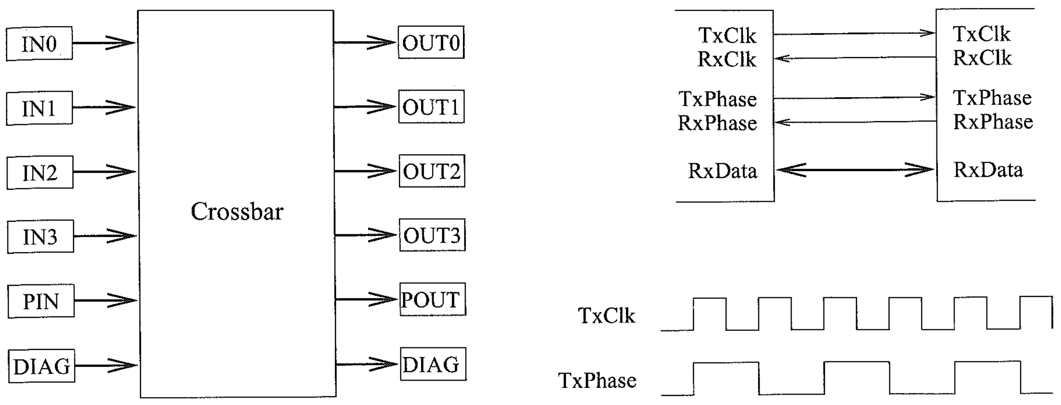


Figure 7.23. Block diagram of the Reliable Router.

used for byte parity bits (BP), kind of flit (Kind), virtual channel identification (VCI), flow control to implement the unique token protocol (Copied Kind, Copied VCI, Freed), to communicate link status information (U/D, PE), and two user bits (USR1, USR0). For a given direction of transfer, the clock is transmitted along with the four data frames as illustrated in the figure. Data are driven on both edges of the clock. To enable the receiver to distinguish between the four frames and reassemble them into a flit, the transmitting side of the channel also sends a pulse on the *TxPhase* signal which has the relative timing as shown. The flit is then assembled and presented to the routing logic. This reassembly process takes two cycles. Each router runs off a locally generated 100 MHz clock removing the problems with distributing a single global clock. Reassembled flits pass through a synchronization module for transferring flits from the transmit clock domain to the receive clock domain with a worst-case penalty of one cycle (see [76] for a detailed description of the data synchronization protocol). The aggregate physical bandwidth in one direction across the channel is 3.2 Gbits/s.

Bit Field	63:12	11	10	9:5	4:0
Contents	User info	Priority	Diagnostic	Address in Y	Address in X

Figure 7.24. Reliable Router head flit format.

Bit Field	22	21	20:18	17	16	15:0
Frame 0	PE	USR0	VCI	BP1	BP0	Data[15:0]
Frame 1	Copied kind		Copied VCI	BP3	BP2	Data[31:16]
Frame 2	U/D	USR1	Kind	BP5	BP4	Data[47:32]
Frame 3	Freed			BP7	BP6	Data[63:48]

Figure 7.25. Reliable Router frame format for data flits.

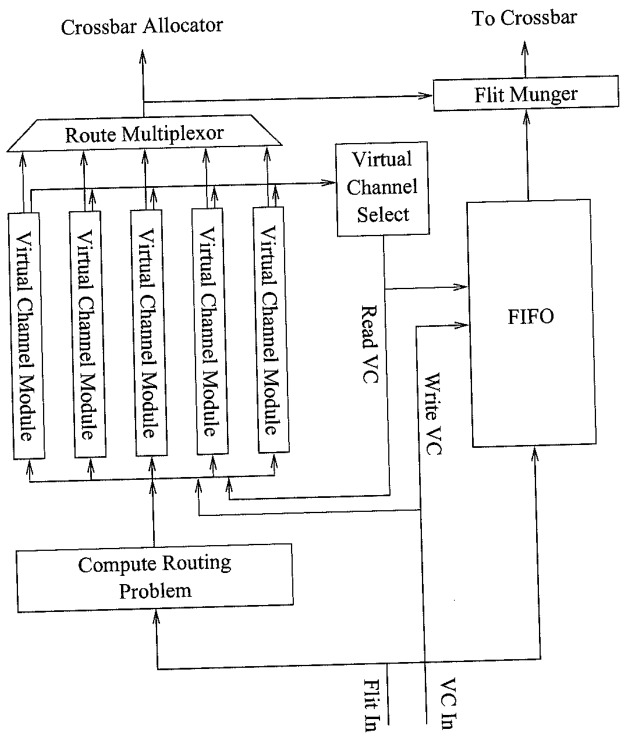


Figure 7.26. Block diagram of the input controller.

While the output controllers simply transmit the flit across the channel, generating the appropriate control and error checking signals, the input controllers contain the core functionality of the chip, and are organized as illustrated in Figure 7.26. The crossbar provides switching between the physical input channels and the physical output channels. Each physical channel supports five virtual channels, which share a single crossbar port: two channels for fully adaptive routing, two channels for dimension-ordered routing, and one for fault handling. The fault handling channels utilize turn model routing. The two dimension-ordered channels support two priority levels for user packets. The router is input buffered. The FIFO block is actually partitioned into five distinct FIFO buffers — one corresponding to each virtual channel. Each FIFO buffer is 16 flits deep. Bandwidth allocation is demand driven: only allocated channels with data to be transmitted or channels with new requests (i.e., head flits) compete for bandwidth. Virtual channels with message packets are accorded access to the crossbar bandwidth in a round robin fashion by the scheduler in the Virtual Channel Controller. Data flits simply flow through the virtual channel FIFO buffers, and the virtual channel number is appended to the flit prior to the transmission of the flit through the crossbar and across the physical channel to the next router. Flow control guarantees the presence of buffer space. When buffers at the adjacent node are full, the corresponding virtual channel does not compete for crossbar bandwidth.

A more involved sequence of operations takes place in routing a head flit. On arrival the head flit is transferred to both the FIFO buffer and a block that compares the destination address with the local node address. The result is the *routing problem* which contains all of the information used to route the message. This information is stored in the virtual channel module, along with routing information: the address of the output controller and the virtual channel ID. Recall from the protocol description provided in Section 6.7.2, the header information must be retained in the router to construct additional messages in the presence of a fault. To minimize the time spent in arbitration for various shared resources, the routing logic is replicated within each virtual channel module. This eliminates the serialized access to the routing logic and only leaves arbitration for the output controllers which is handled by the crossbar allocator. The routing logic first attempts to route a message along an adaptive channel, failing which a dimension-order channel is requested. If the packet fails in arbitration, on the next flit cycle, the router again first attempts an adaptive channel. The virtual channel module uses counters and status signals from adjacent nodes (see *Freed* bits in Figure 7.25) to keep track of buffer availability in adjacent routers. A virtual channel module is eligible to bid for access to the crossbar output only if the channel is routed, buffer space is available on the adjacent node, and there is data to be transmitted. When a fault occurs during transmission, the channel switches to a state where computation for retransmission occurs. After a period of two cycles, the channel switches back to a regular state and competes for access to the crossbar outputs. A header can be routed and the crossbar allocated in two cycles, i.e., 20 ns. The worst case latency through the router is projected to be eight cycles or 80 ns.

Arbitration is limited to the output of the crossbar, and is resolved via three packet priority levels: two user priority levels, and the highest level reserved for system packets. Starvation is prevented by changing the packet priority to level 3 after an input controller has failed to gain access to an output controller after seven tries. Packet selection within a priority level is randomized.

The implementation of the unique token protocol for reliable transmission necessitates some special handling and architectural support. The token at the end of a message must be forwarded only after the corresponding flit queue has successfully emptied. Retransmission on failure involves generation of duplicate tokens. Finally, flow control must span two routers to ensure that a duplicate copy of each data flit is maintained at all times in adjacent routers. When a data flit is successfully transmitted across a channel, the copy of the data flit two routers upstream must be deallocated. The relevant flow control signals are passed through frames as captured in Figure 7.25.

SGI SPIDER

The SGI SPIDER (Scalable Pipelined Interconnect for Distributed Endpoint Routing) [118] was designed with multiple missions in mind: as part of a conventional multiprocessor switch fabric, as a building block for large-scale nonblocking central switches, and as a high-bandwidth communication fabric for distributed graphics applications. The physical links are full-duplex channels with 20 data bits, a frame signal, and a differential clock signal in each direction. Data are transferred on both edges of a 200 MHz clock realizing a raw data rate of 1 Gbyte/s in each direction. The chip core operates at 100 MHz, providing 80-bit units that are serialized into 20-bit phits for transmission over the channel. The organization of the SPIDER chip is shown in Figure 7.27.

The router implements four 256-byte virtual channels over each physical link. The units of buffer management and transmission are referred to as *micropackets*, equivalent to the notion of a flit. Each micropacket is comprised of 128 bits of data, 8 bits of sequencing information, 8 bits of virtual channel flow control information, and 16 CRC bits to protect against errors. The links implement automatic retransmission on errors until the sequence number of the transmitted flit is acknowledged. The micropacket format is shown in Figure 7.28. The number on top of each field indicates the field size in bits. Virtual channel buffer management utilizes credit-based flow control. On initialization, all channels credit their counterparts with buffer space corresponding to their size.

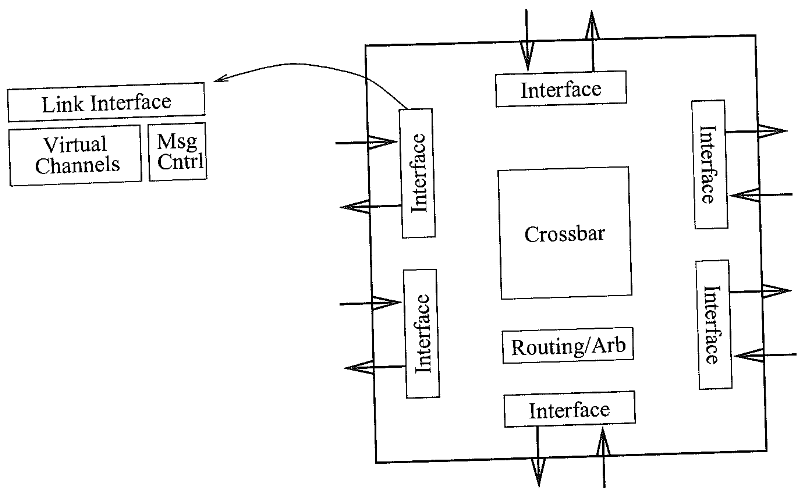


Figure 7.27. Organization of the SGI SPIDER chip.

The 8 bits of virtual channel flow control information are used to identify the virtual channel number of the transmitted micropacket, the address of the virtual channel being credited with additional buffer space, and the amount of credit. An interesting feature of SPIDER links is the transparent negotiation across the link to determine the width of the channel. For example, the router can negotiate to use only the lower 10 bits of the channel. The interface to the chip core remains the same, while the data rate will be a function of the negotiated port width.

The SPIDER chip supports two styles of routing. The first is source routing which is referred to as *vector routing*. This mechanism is utilized for network configuration information and administration. The second style is table-driven routing. The organization of the tables is based on a structured view of the network as a set of metadomains, and local networks within each domain. The topology of the interdomain network and the local network can be different. Messages are first routed to the correct destination domain and then to the correct node within a domain. Routing is organized as shown in Figure 7.29. The destination address is a 9-bit field, comprised of two subfields: a 5-bit metadomain field and a 4-bit local field. The tables map these indices into 4-bit router port addresses, thereby supporting routers with up to 16 ports. Access to the metatable provides a port address to route the message toward the correct domain. If the message is already in the correct domain as determined by comparison with the local meta ID, the local address table is used to determine the output port that will take the message towards the destination node. Not shown in the figure is a mode to force

8	4	4	64
Flow control	Seq #	Ack Seq #	Data
Check bits			Data

Figure 7.28. SPIDER data micropacket format.

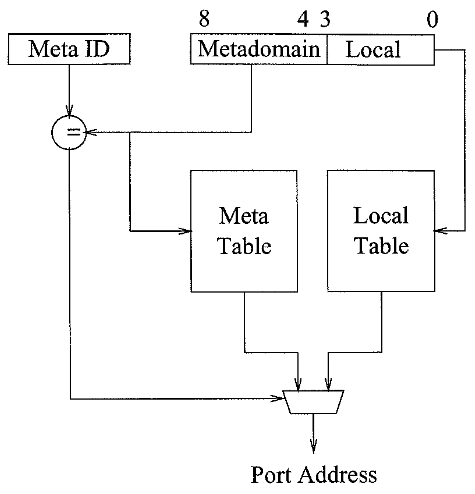


Figure 7.29. SPIDER routing tables.

the message to use the local address. Table-driven routing is oblivious and forces messages to use fixed paths. The table entries are computed to avoid deadlock and can be reinitialized in response to faults. Messages using vector routing can be used to reload the tables.

Normally arbitration for the output of the crossbar is not possible until the routing operation has been completed and has returned a crossbar output port address. In order to overlap arbitration and table lookup, the routing operation described above produces the output port address at the next router. This 4-bit port address (Dir) is passed along with the message. At the adjacent router, the port address is extracted from the header and can be immediately submitted for crossbar output arbitration while table lookup progresses in parallel. This overlap saves one cycle. The header is encoded into the 128 data bits of a micropacket. The overall header organization is shown in Figure 7.30. The number on top of each field indicates the field size in bits. The age field establishes message priorities for arbitration (see below) and the CC (congestion control) field permits the use of multiple virtual channels.

To maximize crossbar bandwidth each input virtual channel buffer is organized as a set of linked lists of messages, one for each output port (five in this chip). This organization prevents messages blocked on an output port from blocking other messages in the same buffer destined for a different output port. With 24 input virtual channels, there are potentially 120 candidates for arbitration. To avoid starvation, messages in the router are aged at a programmable rate and arbitration is priority-driven. The top 16 priority levels of a total of 255 levels are reserved.

41	9	4	8	2
Data	Dest ID	Dir	Age	CC
Data				

Figure 7.30. SPIDER header micropacket format.

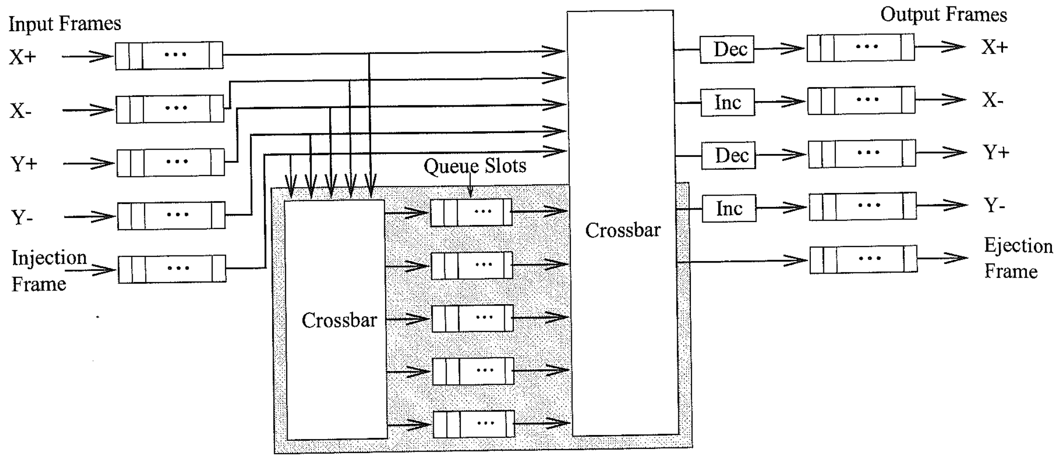


Figure 7.31. Block diagram of the Chaos router. (Dec = Decrementer; Inc = Incrementer.)

Support is provided within the chip to track message statistics such as retransmission rates on links so that problematic links can be identified and may be shut down. Timers support the detection of error conditions and potential deadlock situations, e.g., those due to errors. Timer expiration may result in either micropackets injected into the message stream for message recovery, or in resetting of a router port. A real-time clock is also distributed through the network to support tight synchronization. The pad-to-pad no-load latency through the chip is is estimated to be 40 ns, while the latency from input on a router to the input on the adjacent router is estimated to be 50 ns.

7.2.4 VCT Routers

Routers supporting VCT switching share many of the attributes of wormhole routers. The principal distinguishing feature is the availability of sufficient space for buffering complete message packets. As a result, deadlock freedom is generally achieved through buffer management rather than routing restrictions. The following examples discuss three router architectures currently under development.

The Chaos Router

The Chaos router chip is an example of a router designed for VCT switching for operation in a 2-D mesh. To reduce pin count, each channel is a 16-bit, half-duplex bidirectional channel. The chip was designed for a cycle time of 15 ns. The latency for the common case of cut-through routing (no misrouting) is four cycles from input to output. This performance compares favorably to that found in more compact oblivious routers, and has been achieved by keeping the misrouting and buffering logic off of the critical path. A block diagram of the Chaos router is shown in Figure 7.31.

The router is comprised of the router core and the multiqueue [32]. The core connects *input frames* to *output frames* through a crossbar switch. Each frame can store one 20-flit message packet. Each flit is 16 bits wide and corresponds to the width of the physical channel. The architecture of the core is very similar to that of contemporary oblivious routers, with the additional ability to buffer complete 20-flit packets in each input and output frame. Under low load conditions, the majority of the traffic flows through this core, and therefore considerable attention was paid to the optimization of the performance of the core datapath and control flow. However, when the network starts to become congested, messages may be moved from input frames to the multiqueue, and subsequently from the multiqueue to an output frame. When a packet is stalled in an input frame waiting for an output frame to become free, this packet is moved to the multiqueue to free channel resources to be used by other messages. To prevent deadlock, packets may also be moved from input frames to the multiqueue if the output frame on the same channel has a packet to be transmitted. This ensures that both receiving and transmitting buffers on both sides of a channel are not occupied, ensuring progress (reference the discussion of chaotic routing in Chapter 6) and avoiding deadlocked configurations of messages. Thus, messages may be transmitted to output frames either from an input frame or from the multiqueue, with the latter having priority. If the multiqueue is full and a message must be inserted, a randomly chosen message is misrouted.

Although several messages may be arriving simultaneously, the routing decisions are serialized by traversing the output channels in order. The *action dimension* is the current dimension under consideration (both positive and negative directions are distinguished). Packets in the multiqueue have priority in being routed to the output frame. If a packet in the multiqueue requires the active dimension output and the output frame is empty, the packet is moved to the output frame. If the queue is full and the output frame is empty, a randomly selected packet is misrouted to this output frame. Misrouting is the longest operation and takes five cycles. If the multiqueue operations do not generate movement, then any packet in an input frame requesting this output dimension can be switched through the crossbar. This operation takes a single cycle. The routing delay experienced by a header is two cycles. Finally, if the input frame of the active dimension is full, it is read into the multiqueue. This implements the policy of handling stalled packets and the deadlock avoidance protocol. The preceding steps encompass the process of routing the message. The no-load latency through the router is four cycles: one cycle for the header to arrive on the input, two cycles for the routing decision, and one cycle to be switched through the crossbar to the output frame [32]. If a message is misrouted, the routing decision takes five cycles.

Logically, the multiqueue contains messages partially ordered according to the output channel that the messages are waiting to traverse. The above routing procedure requires complete knowledge of the contents of the multiqueue and relative age of the messages. The current implementation realizes a five-packet multiqueue. This logically corresponds to five queues: one for each output including the output to the processor. Since messages are adaptively routed, a message may wait on one of several output channels, and therefore must logically be entered in more than one queue. When a free output frame is available for transfers from the multiqueue, determination of the oldest message waiting on this output must be readily available. The state of the multiqueue is maintained in a destination scoreboard.

The logical structure of the destination scoreboard is illustrated in Figure 7.32 for a multiqueue with six slots. This scoreboard keeps track of the relationship between the messages in the multiqueue, their FIFO order, and the destination outputs. Each row corresponds to an output frame in a particular dimension or to the local processor. Each column corresponds to one of the five slots in the multiqueue. When a packet enters the tail of the queue, its status enters from the left, i.e., in the first column. In this column, each row entry is set to signify that the corresponding output channel is a candidate output channel for this packet. For example, the packet that is located at the tail of the queue can be transmitted out along the $Y+$ or $X-$ directions, whichever becomes available. As packets exit the multiqueue, the columns in the scoreboard are shifted right. When an output frame is freed, the first message destined for that frame is dequeued from the central queue. Thus, messages may not necessarily be removed from the head of the queue. In this case some column in the middle of the matrix is reset, and all columns are compressed right corresponding to

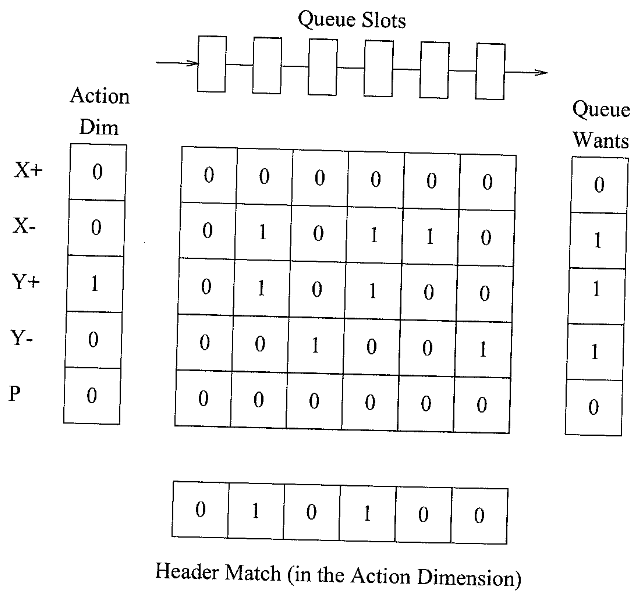


Figure 7.32. Tracking messages in the Chaos multiqueue.

the compression in the queue when a message is removed from the middle. It is apparent that taking the logical OR of row elements indicates if there is a message in the queue for the corresponding output frame. Given a specific dimension (e.g., the action dimension) it is apparent that row and column logical operations will produce a vector indicating messages that are queued for the action dimension. The relative positions of the bits set in this vector determines the relative age of the multiple messages and is used to select the oldest. Thus, all messages are queued in FIFO order, while messages destined for a particular output are considered in the order that they appear in this FIFO buffer.

The maximum routing delay for a message packet is five cycles, and occurs when messages are misrouted. This delay can be experienced for each of the four outputs in a 2-D mesh (the processor channel is excluded). Since routing is serialized by output, each output will be visited at least once every 20 cycles. Thus, messages are 20 flits to maintain maximum throughput. In reality, the bidirectional channels add another factor of 2, therefore in the worst case we have 40 cycles between routing operations on a channel.

Finally, the router includes a number of features in support of fault-tolerant routing. Timers on the channels are used to identify adjacent faulty channels/nodes. Packets bodies are protected by checksums while the header portion is protected by parity checks. A header that fails a parity check at an intermediate node, is ejected and handled in software. A separate single bit signal on each channel is used to propagate a fault detection signal throughout the network. When a router receives this signal, message injections are halted and the network is allowed to "drain" [35]. Messages which do not successfully drain from the network are detected using a timer and ejected to the local processor. When all messages have been removed from the faulty network, the system enters a diagnostic and recovery phase. Recovery is realized by masking out faulty links. This is achieved within the routing decision logic by performing the logical AND of a functional channel

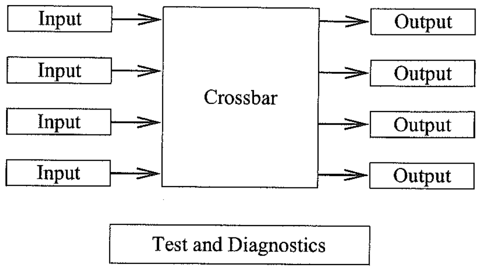


Figure 7.33. The Arctic routing chip.

mask with a vector of candidate output channels computed from the routing header. This will only permit consideration of nonfaulty output channels. If the result indicates that no candidate channels exist (this may occur due to faults), the packet is immediately misrouted.

Currently an active project is underway to implement a version of the Chaos router as a LAN switch. This switch will be constructed using a new version of the Chaos router chip and will utilize 72-byte packets and full-duplex, bidirectional channels with 8-bit wide links. As described in Section 7.2.2, the arbitration between two sides of a half-duplex pipelined channel carries substantial performance penalties. With LAN links being potentially quite long, half-duplex links represented a poor design choice. The projected speed of operation is 180 Mhz using the byte-wide channels for a bidirectional channel bandwidth of 360 Mbytes/s. The LAN switch will be constructed using a network of Chaos router chips. For example, a 16-port switch can be constructed using a 4×4 torus network of Chaos routers. Routing internal to the switch is expected to be normal Chaos routing while switch to switch routing is expected to use source routing and standard FIFO channels with link pipelining, if the links are long enough. Packet-level acknowledgments will be used for flow control across the LAN links. With the design of a special-purpose interface, the goal is to obtain host-to-host latencies down in the 10–20 μs range. This new-generation chip design is being streamlined with clock cycle times approaching 5.6 ns (simulated). This would produce a minimal delay through the router of four cycles or 22.4 ns. Packet formats and channel control have also been smoothed in this next generation chip to streamline the message pipeline implementation.

Arctic Router

The Arctic routing chip is a flexible router architecture that is targeted for use in several types of networks, including the fat tree network used in the *T multiprocessor [269]. The basic organization of the router is illustrated in Figure 7.33, and the current prototype is being used as part of a network with PowerPC compute nodes. Four input *sections* can be switched to any one of four output sections. The router is input-buffered and each input section is comprised of three buffers. Each buffer is capable of storing one maximum sized packet of 96 bytes in support of VCT switching. The crossbar switch connects the 12 inputs to four outputs. Message packets have one of two priority levels. The buffers within an input unit are managed such that the last empty buffer is reserved for a priority packet.

The physical channels are 16-bit full-duplex links operating at 50 MHz, providing a bandwidth of 200 Mbytes/s for each link. The internal router datapaths are 32 bits wide. The Arctic physical channel interface includes a clock and frame signal as well as a flow control signal in the reverse direction. The flow control is used to enable counter increment/decrement in the transmitter to keep track of buffer availability on the other side of the channel. The initial value of this counter is configurable enabling the chip to interface with other routers that support different buffer sizes.

Routing is source-based, and networks can be configured for fat trees, Omega, and grid topologies. In general, two subfields of the header (perhaps single bits) are compared with two reference values. Based on the results of this comparison, 2 bits are selected from the packet header, or some combination of header and constant 0/1 bits are used to produce a 2-bit number to select the output channel. For the fat tree routing implementation, 8 bytes of the message are devoted routing, control, and CRC. The routing header is comprised of two fields with a maximum width of 30 bits: 14 bits for the path up the tree, and 16 bits for the path down the tree to the destination. Since routing is source-based, and multiple alternative paths exist up the tree, the first field is a random string. Portions of this field are used to randomize the choice of the up path. Otherwise a portion of the second field is used to select an appropriate output headed down the tree to the destination. Rather than randomizing all of the bits in the first field, by fixing the values of some of the bits such as the first few bits, messages can be steered through regions of the lower levels of the network. Source based routing can be utilized in a very flexible way to enforce some load balancing within the network. For example, I/O traffic could be steered through distinct parts of a network to minimize interference with data traffic. Messages traversing the Arctic fabric experience a minimum latency of six cycles/hop.

The Arctic chip devotes substantial circuitry to test and diagnostics. CRC checks are performed on each packet, and channel control signals are encoded. Idle patterns are continuously transmitted when channels are not in use to support error detection. On-chip circuitry and external interfaces support static configuration in response to errors: ports can be selectively disabled and flushed and the route tables used for source routing can be recomputed to avoid faulty regions. An interesting feature of the chip is the presence of counters to record message counts for analysis of the overall network behavior.

R2 Router

The R2 router evolved as a router for a second-generation system in support of low-latency high-performance interprocessor communication. The current network prototype provides for interprocessor communication between Hewlett-Packard commercial workstations. The R2 interconnect topology is a 2-D hexagonal interconnect and the current router is designed to support a system of 91 nodes. Each router has seven ports: six to neighboring routers and one to a local network interface (referred to as the fabric interface component or FIC). The topology is wrapped, i.e., routers at the edge of the network have wraparound links to routers at the opposite end of the dimension. The network topology is shown in Figure 7.34. For clarity, only one wrapped link is illustrated. As in multidimensional tori, the network topology provides multiple shortest paths between each pair of routers. However, unlike tori, there are also two *no-farther* paths. When a message traverses a link along a no-farther path, the distance to the destination is neither increased nor decreased. The result is an increase in the number of routing options available at an intermediate router.

The physical channel is organized as a half-duplex channel with 16 bits for data and 9 bits for control. One side of each link is a router port with an arbiter which governs channel mastership. The routers and channels are fully self-timed, eliminating the need for a global clock. There are no local clocks and resynchronization with the local node processor clocks is performed in the network interface. The rate at which the R2 ports can toggle is estimated to be approximately 5 – 10 ns corresponding to an equivalent synchronous transmission rate of 100 – 200 MHz.

Message packets may be of variable length and upto a maximum of 160 bytes. Each packet has a 3-byte header that contains addressing information, and a 16-byte protocol header that contains

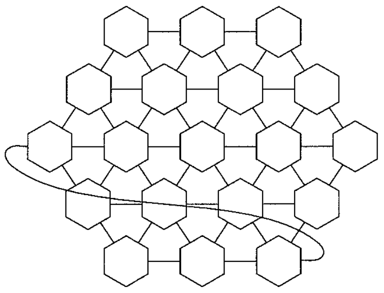


Figure 7.34. Wrapped hexagonal interconnect used in the R2-based network.

information necessary to implement a novel *sender-based protocol* for low-latency messaging. A 4-byte CRC trailer rounds out the message packet format. Two types of messages are supported: normal messages and priority messages. While normal messages may be as large as 64K packets in length, priority messages are limited to a single packet. Typically priority messages are used by the operating system while normal messages are generated by the applications.

A block diagram of the R2 router is illustrated in Figure 7.35. In the figure the ports are drawn as distinct inputs and outputs to emphasize the data path, although they are bidirectional. Each FIFO buffer is capable of storing one maximum-length packet. There are 12 FIFO buffers for normal packets and three FIFO buffers for storing priority packets. Two 7 × 15 crossbars provide switching between the buffers and the bidirectional ports. Each of the FIFO buffers has a routing controller. Routing decisions use the destination address in the packet header and the local node ID. Buffer management and ordered use of buffers provides deadlock freedom. The implementation is self-timed, and the time to receive a packet header and to compute the address of an output port is estimated to take between 60 and 120 ns.

Routing is controlled at the sender by specifying a number of *adaptive credits*. Each adaptive credit permits the message packet to traverse one link that takes the packet no further from the

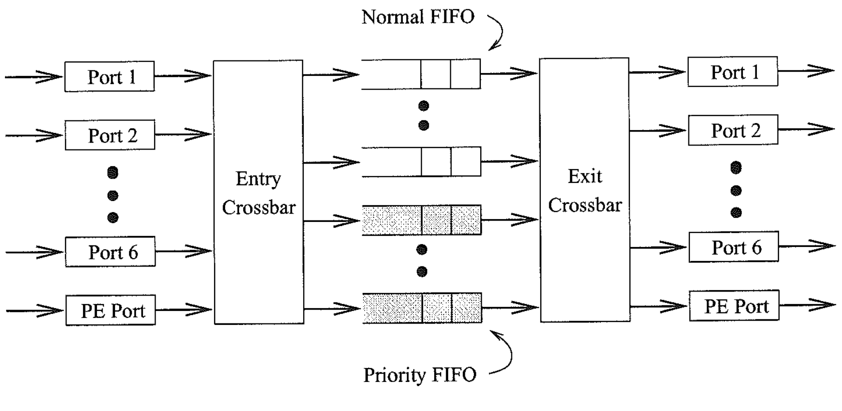


Figure 7.35. Organization of the R2 router.

destination. At any node, there are always two no-farther directions. The adaptive credit field in the header is decremented each time a message takes an adaptive step and further adaptive routing is not permitted once this value is reduced to zero. Shortest paths are given priority in packet routing. Priority packets are only routed along shortest paths. When all outputs for a priority packet are blocked by a normal packet, the priority packet is embedded in the normal packet byte stream being transmitted along one of the shortest paths, analogous to the implementation of virtual channels. An interesting feature of the R2 router is the support for message throttling. Packet headers are monitored and in the presence of congestion, rapid injection of packets from a source can cause negative acknowledgments to be propagated back to the source to throttle injection.

Error handling was considered at the outset during the initial design, and basic mechanisms were provided to detect packets which could not be delivered. Once detected, these packets were ejected at the local node to be handled in software by higher-level protocols. A variety of conditions may cause message exceptions. For example, the message packet headers contain a field that keeps track of the maximum number of hops. This field is decremented at each intermediate router, and when this value becomes negative, the message is ejected. An ejection field in the header is used to mark packets that must be forwarded by the router to the local processor interface. The router also contains watchdog timers that monitor the ports/links. If detection protocols time out, a fault register marks the port/link as faulty. If, the faulty port leads to a packet destination, this status is recorded in the header and the packet is forwarded to another router adjacent to the destination. If the second port/link to the destination is also found to be faulty, the packet is ejected.

7.2.5 Circuit Switching

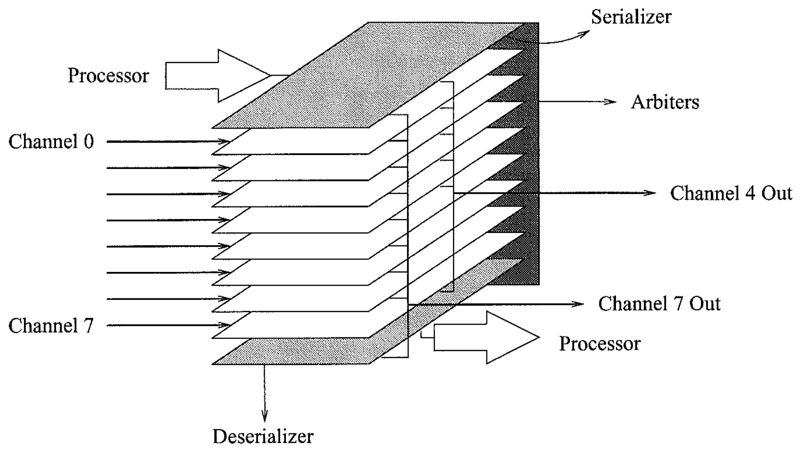
Circuit switching along with packet switching represent the first switching techniques used in multiprocessor architectures. This section presents one such example and some more recent ideas toward improving the performance of the next generation of circuit-switched networks.

Intel iPSC Direct Connect Module (DCM)

The second generation network from Intel included in the iPSC/2 series machines utilized a router that implements circuit switching. Although most machines currently use some form of cut-through routing, circuit-switched networks do present some advantages. For example, once a path is setup, data can be transmitted at the full physical bandwidth without delays due sharing of links. The system can also operate completely synchronously without the need for flow control buffering of data between adjacent routers. While the previous generation of circuit-switched networks may have been limited by the path lengths, the use of link pipelining offers new opportunities.

The network topology used in the iPSC series machines is that of a binary hypercube. The router is implemented in a module referred to as the *direct connect module* (DCM). The DCM implements dimension-ordered routing and supports up to a maximum of eight dimensions. One router port is reserved for direct connections to external devices. The remaining router ports support topologies of up to 128 nodes. The node architecture and physical channel structure is shown in Figure 7.36. Individual links are bit-serial, full-duplex, and provide a link bandwidth of 2.8 Mbytes/s. Data, control, and status information are multiplexed across the serial data link. The strobe lines supply the clock from the source. The synchronous nature of the channel precludes the need for any handshaking signals between routers.

When a message is ready to be transmitted, a 32-bit word is transferred to the DCM. The least significant byte contains the routing tag — the exclusive-OR of the source and destination addresses. These bits are examined in ascending order by each routing element. A bit value of 1 signifies that



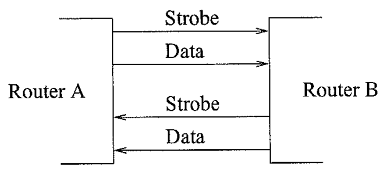


Figure 7.36. The iPSC/2 circuit switched router organization.

the message must traverse the corresponding dimension. The input Serializer generates a request for the channel in the dimension corresponding to the least significant asserted bit in the routing tag. Subsequent routing elements at intermediate nodes receive the header, and make similar requests to output channels. Multiple requests for the same output channel are arbitrated and resolved in a round-robin fashion. Note the connectivity at the outputs in Figure 7.36. An outgoing channel can expect to receive a message only from lower-numbered channels. This significantly reduces the design complexity of the routing elements. Channel status information is continually multiplexed across the data lines in the reverse direction. The status information corresponds to *RDY* or *EOM* (end of message) signals. These signals achieve end-to-end flow control. For example, when the destination DCM receives a routing header, the *RDY* status signal transmitted back to the source DCM initiates the actual transfer of data. The *EOM* status signal appended to the message causes the reserved links to be freed as it traverses the path. The propagation of status signals through the DCM from input to output is achieved by keeping track of the channel grants generated during the setup of the original message path. This end-to-end flow control also permits the propagation of "not ready" status information to the source to enable throttling of message injections.

Optimizing Circuit Switching

Communication requirements heavily depend on the application running on the parallel machine and the mapping scheme used. Many applications exhibit both spatial and temporal communication locality. If the router architecture supports circuit switching, conceivably compile time analysis could permit the generation of instructions to set up and retain a path or circuit that will be heavily used over a certain period of time. It is possible to design a router architecture such that a circuit is set up and left open for future transmission of data items. This technique was proposed in [39] for systolic communication. It has also been proposed in [159] for message passing and extended in [82]. The underlying idea behind preestablished circuits is similar to the use of cache memory in a processor: a set of channels is reserved once and used several times to transmit messages.

In structured message-passing programs, it may be feasible to identify and set up circuits prior to actual use. This set up may be overlapped with useful computation. When data are available, the circuit would have already been established, and permit lower latency message delivery. However, setting up a circuit in advance only eliminates some source software overheads such as buffer allocation, as well as routing time and contention. This may be a relatively small advantage compared to the bandwidth wasted by channels that have been reserved but are not currently used. Circuits could be really useful if they performed like caches, i.e., preestablished circuits actually operate faster. Link-level pipelining can be employed in a rather unique way to ensure this cache-like behavior. We use the term *wave pipelining* [102] to represent data transmission along such high-speed circuits.

It is possible to use wave pipelining across switches and physical channels, enabling very high clock rates. With a preestablished circuit, careful design is required to minimize the skew between wires in a wave-pipelined parallel data path. Synchronizers are required at each delivery channel. Synchronizers may also be required at each switch input to reduce the skew. Clock frequency is limited by delivery bandwidth, by signal skew, and by latch setup time. With a proper router and memory design, this frequency can potentially be higher than that employed in current routers, increasing channel bandwidth and network throughput accordingly. Spice simulations [102] show that up to a factor of four increase in clock speed is feasible. Implementations would be necessary to validate the models. The use of pipelined channels allows the designer to compute clock frequency independently of wire delay, limited by routing delay and switch delay. An example of a router organization that permits the use of higher clock frequencies based on the use of wave pipelining across both switches and channels is shown in Figure 7.37.

Each physical channel in switch S_0 is split into $k+w$ virtual channels. Among them, k channels are the control channels associated with the corresponding physical channels in switches $S_1, \ldots, S_k$. Control channels are only used to set up and tear down physical circuits. These channels have capacity for a single flit and only transmit control flits. Control channels are set up using pipelined circuit switching and handled by the PCS routing control unit. The remaining w virtual channels are used to transmit messages using wormhole switching and require deeper buffers. They are handled by the wormhole routing control unit. The hybrid switching technique implemented by this router architecture was referred to as *wave switching* [102]. With such a router architecture, it is possible to conceive of approaches to optimizing the allocation of link bandwidth in support of real-time communication, priority traffic, or high throughput.

7.2.6 Pipelined Circuit Switching

The implementation of routing protocols based on pipelined circuit switching present different challenges from those presented by wormhole-switched routers, notably in the need to provide support for backtracking and the distinction between control flit traffic and data flit traffic. Ariadne [7] is a router for PCS supporting the MB-m family of fault-tolerant routing protocols. The current implementation is for a k-ary 2-cube with 8-bit-wide, half-duplex physical data channels. There are two virtual data channels and one virtual control channel in each direction across each physical link. The router is fully asynchronous and is comprised of three major components — the control path, the physical link interface, and the data path.

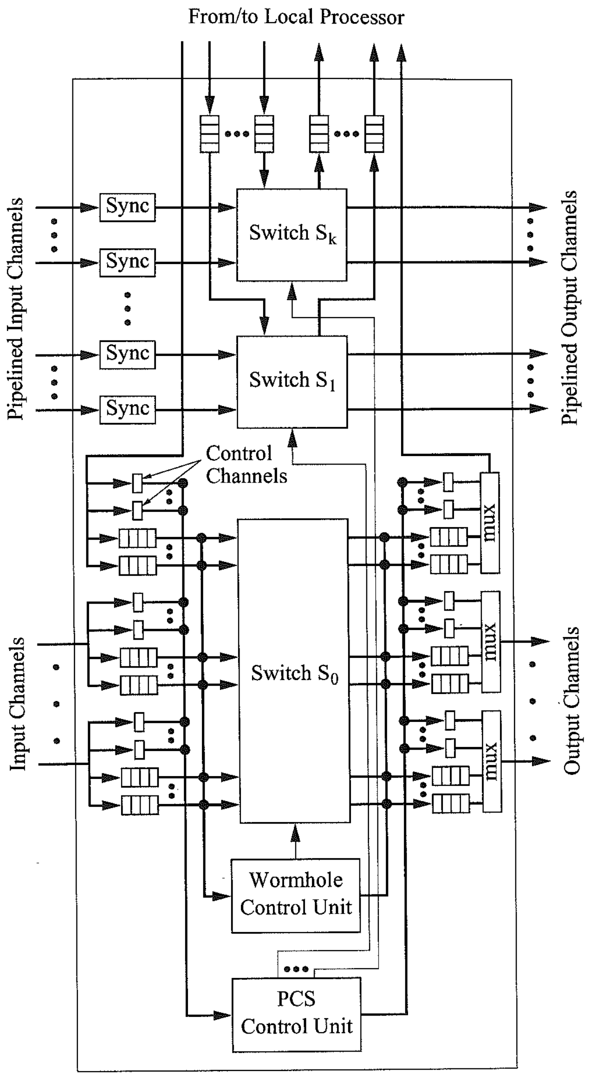


Figure 7.37. Organization of a router employing wave pipelining. (Mux = Multiplexer; PCS = Pipelined circuit switching; Sync = Synchronizer.)

Figure 7.38 illustrates the block diagram of the architecture of the Ariadne chip. Each physical link has an associated link control unit (LCU). These LCUs are split in the diagram with the input half on the left and the output half on the right. Each input LCU feeds a data input buffer unit (DIBU) and a control input buffer unit (CIBU). The buffer units contain FIFO buffers for each virtual channel over the link. The data FIFO buffers feed directly into distinct inputs of the 9×9 crossbar.

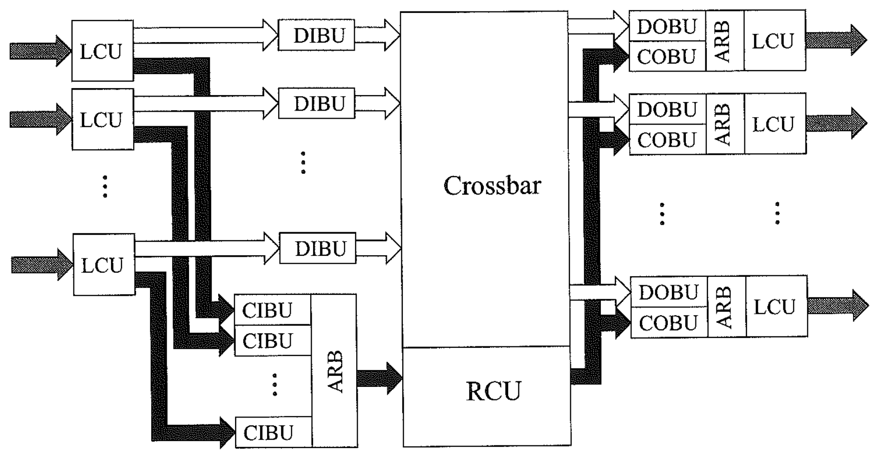


Figure 7.38. Block diagram of Ariadne: a PCS router.

The internal control path is indicated by the solid arrows. Control and data flits arrive over the physical link and are differentiated by the LCU. Control flits are first directed to the CIBUs. The CIBUs send their flits to the router control unit (RCU) via a common bus. The CIBUs arbitrate for use of the bus via a distributed, mutual exclusion mechanism. Once a CIBU has acquired the privilege to use the bus, it presents its routing header and virtual channel address. When the RCU receives a routing header, a routing decision is made, a crossbar mapping is modified, and the header is directed to a control output buffer unit (COBU) via a common output bus.

Figure 7.39 illustrates the signals that comprise each physical link. There are four pairs of request (R) and acknowledge (A) lines. Since the links are half-duplex, nodes must arbitrate for the privilege to send data over the link. One request/acknowledge pair for each side is used in the arbitration protocol (R_{ARBn}, A_{ARBn}). Once a node has the privilege to send, the other request/acknowledge pair is used in the data transfer (R_{DATAn}, A_{DATAn}). An *Accept* line is used to indicate whether or not the flit was accepted by the receiving node or rejected due to input buffer overflow. While there are more pin-efficient methods of implementing these internode handshaking protocols [193], the use of a self-timed design and new switching technique led to the adoption of a more conservative approach.

Virtual channels are granted access to the physical channel using a demand-driven mutual exclusion ring. A signal representing the privilege to drive the physical channel circulates around this ring. Each virtual channel has a station on the ring and will take the privilege if the channel has data to send. In addition, there is also a station on the ring that represents the routing node on the other side of the link (these are half-duplex channels). If the other node has data to send, the privilege will be taken, and the other node will be given control of the link. This scheme multiplexes data in a true demand-driven, rotating fashion among all virtual channels in both directions over the physical link.

While PCS requires one control channel for each virtual data channel, Ariadne combines all control channels into one common virtual control channel. Originally done to reduce circuit area, it also reduces the number of stations on the arbitration rings devoted to control flit FIFO buffers. While it may increase the control flit latency to the RCU, the control flit traffic through the network

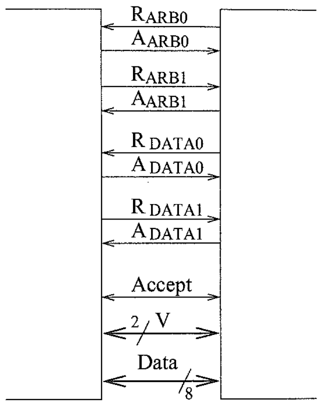


Figure 7.39. Physical link.

is substantially less than the data flit traffic. Each message creates two control flits (the header and acknowledgment flits) and any number of data flits. For most realistic message sizes, it is the data flit population that will be the determinant of message throughput and latency. The larger the message, the smaller the effect of collapsing control channels into a single channel. For example, with 32-flit messages, control flits will represent about 6% of the offered load.

Backtracking protocols must use history information. Rather than carry this information within the routing header, leading to large headers [52, 62], history information is stored locally at a router when necessary. With each virtual data channel we associate a history mask containing a bit for each physical channel. Backtracking protocols use the history mask to record the state of the search at an intermediate node. For example, when a header flit arrives on a virtual channel, say f, the history mask for virtual channel f is initialized to 0, except for the bit corresponding to the physical link of f. The header is routed along some outgoing physical link, say g and the bit for link g is set in the history mask of f. If eventually the header returns to the node backtracking across g, the routing header is prevented from attempting link g again because the bit for g in the history mask of f is set.

The current control flit format is shown in Figure 7.40. All header flits have the Header bit set. Headers will have the Backtrack bit set or cleared depending on whether they are moving forward or backtracking. Acknowledgment flits will have the Header bit cleared and the Backtrack bit set. The next implementation of Ariadne is expected to use 16-bit flits which will allow for larger X and Y offsets as well as an increase in the number of misroutes that are permitted.

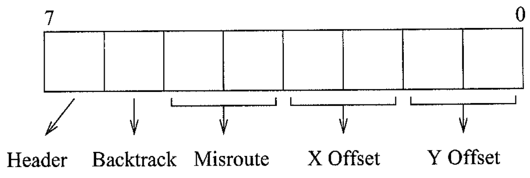


Figure 7.40. Control flit format.

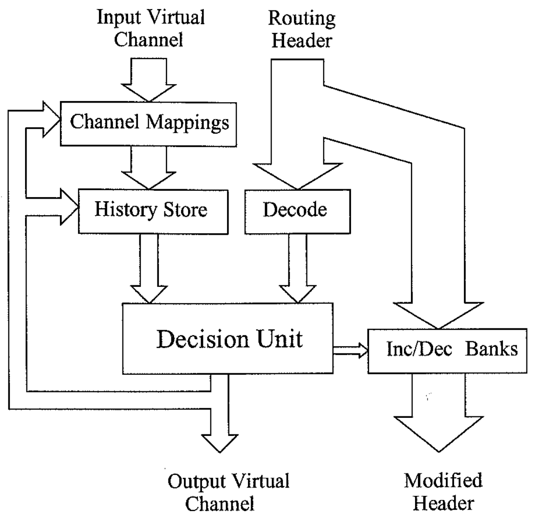


Figure 7.41. Routing control unit.

The routing control unit is more complex than in wormhole-switched routers since history information must be extracted, and the routing protocols are fully adaptive necessitating the selection of one output channel from several candidates. Figure 7.41 shows a block diagram of the RCU. The first step in routing is decoding the routing header and fetching history information. When a forward going routing header enters a node, the history mask is initialized. When a routing header is backtracking, its history mask must be retrieved from the history store. Since the history mask is associated with the virtual channel the header first arrived on, the RCU must first determine the header's point of origination from the crossbar. Once the history information is either initialized or retrieved, a routing decision is made and the values in the outgoing header flit are modified accordingly. The necessary offsets are selected after the routing function has been computed [57]. The selection function favors profitable directions over misrouting.

Data flits are directed to the DIBUs by the LCU. The data input FIFO buffers feed the input ports of a 9×9 crossbar. It is a full virtual-to-virtual channel crossbar — four physical channels with two virtual channels/physical channel and one channel to the processor. The data output FIFO buffers receive their inputs from the output ports of the crossbar. The crossbar is constructed in such a way that the mapped status of every input and output can be read by the RCU. In addition, the RCU may query the crossbar to determine the input that is mapped to an output. Thus, the RCU can obtain the information necessary to direct backtracking headers and acknowledgment flits to their parent node in the path.

Critical sections of the design were simulated using Spice with the level three CMOS transistor models provided by MOSIS for the HP CMOS26B 0.8 μm process. Due to the self-timed nature of the router, the intranode delays incurred by control and data flits is nondeterministic due the arbitration for the RCU and physical links. Since routing headers follow a different path through the router than data flits, the delays are presented in separate tables. The routing header delay is provided in Table 7.5. Adding all of the components, the intranode latency of routing headers ranges from 21 to 48 ns (line 1, Table 7.5). Note that this includes both intranode delay and pad

Table 7.5. Simulated (Spice) timing: Path setup.

Parameter	Time
Control path	13.75 ns + RCU + $2t_{arb}$
RCU (Forward-going header)	13.0 – 16.0 ns
RCU (Acknowledgment)	7.5 ns
RCU (Backtracking header)	16.0 – 19.0 ns

Table 7.6. Simulated (Spice) timing: Data path.

Parameter	Time
Data path	8.5 ns + t_{arb}
Link cycle time (1 active channel)	16.4 ns + t_{synch} + $2t_l$
Link cycle time (>1 active channels)	13.65 ns + $2t_l$
Link idle time for direction change	4.3 ns + t_l

delays as well as channel arbitration delays. Table 7.6 summarizes the major delay components for data flits in Ariadne. In the table, t_{synch} represents the portion of the channel delay that cannot be overlapped with arbitration while t_{arb} represents the worst-case delay in arbitration. Modeled datapath bandwidths are in the vicinity of 50–75 Mbytes/s depending on traffic and message distribution.

7.2.7 Buffered Wormhole Switching

This variant of wormhole switching is employed in the SP-2 interconnection network: a bidirectional multistage interconnection network constructed from 4 × 4 bidirectional buffered crossbar switches. Two stages of four switches each, are organized into a *frame* that supports up to 16 processors. The architecture of a frame is shown in Figure 7.42. Each crossbar switch has four bidirectional inputs and four bidirectional outputs, and therefore operates as an 8 × 8 crossbar. In this two-stage organization, except for processors on the same switch, there are four paths between every pair of processors connected to the same frame. The use of bidirectional links avoids the need for costly (i.e., long) wraparound links from the outputs back to the inputs. Multiple frames can be connected as shown in Figure 7.42 to connect larger numbers of processors, either by directly connecting frames (up to 5 frames or 80 processors) or by cascading frames together. In the cascaded organization, switch frames are used in the last stage to provide wraparound links as shown in Figure 7.42. As the systems grow larger, the number of paths between any pair of processors grows. For example, the 128-node configuration provides 16 paths between any pair of frames.

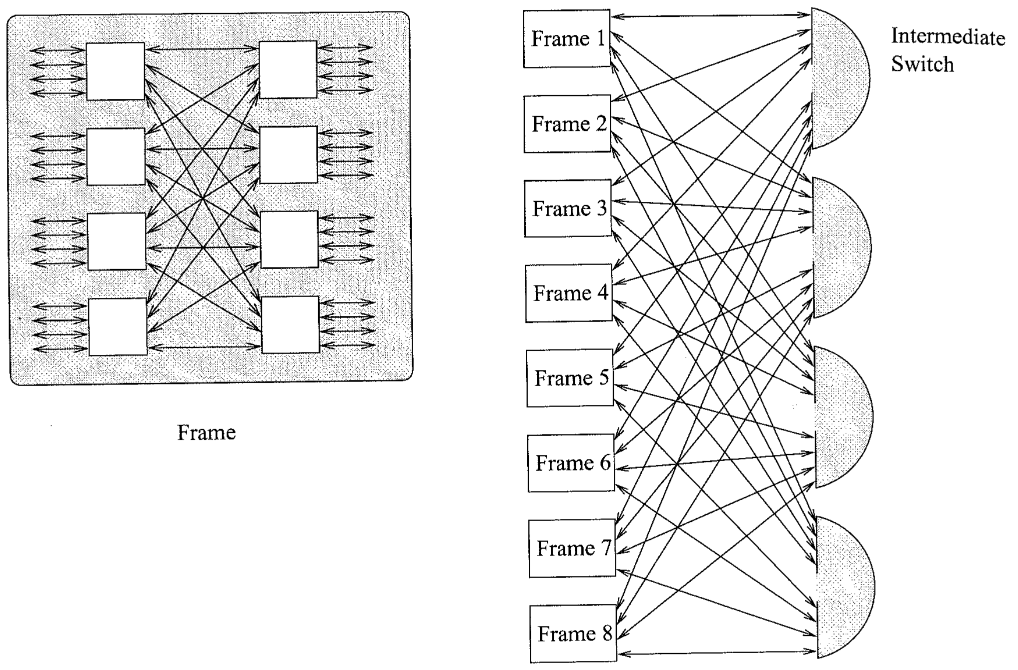


Figure 7.42. System interconnect using an SP-2 frame.

The SP-2 physical channel appears as shown in Figure 7.43. The data channel is 8 bits wide, with a single-bit tag signal and a single-bit token signal in the reverse direction. With the addition of a clock signal, the channel is 11 bits wide. SP-2 flits are matched to the channel width at 8 bits. An active tag signifies valid token on the channel. Tokens are encoded in two cycle frames: a sequence of 0–1 represents no tokens and 1–0 represents two tokens. The token sequence is used to increment a counter in the receiving port. The counter keeps track of the number of flits that can be sent and is decremented each time a flit is transmitted. The other patterns are used for error detection.

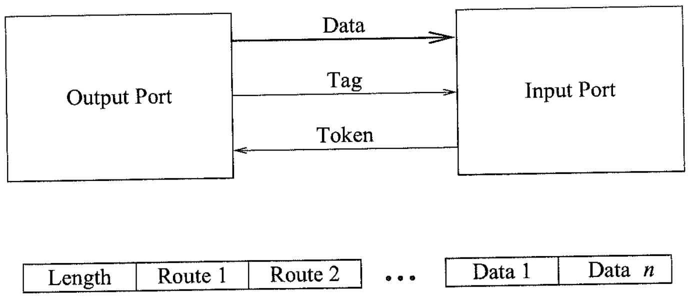


Figure 7.43. SP-2 physical channel and message packet formats.

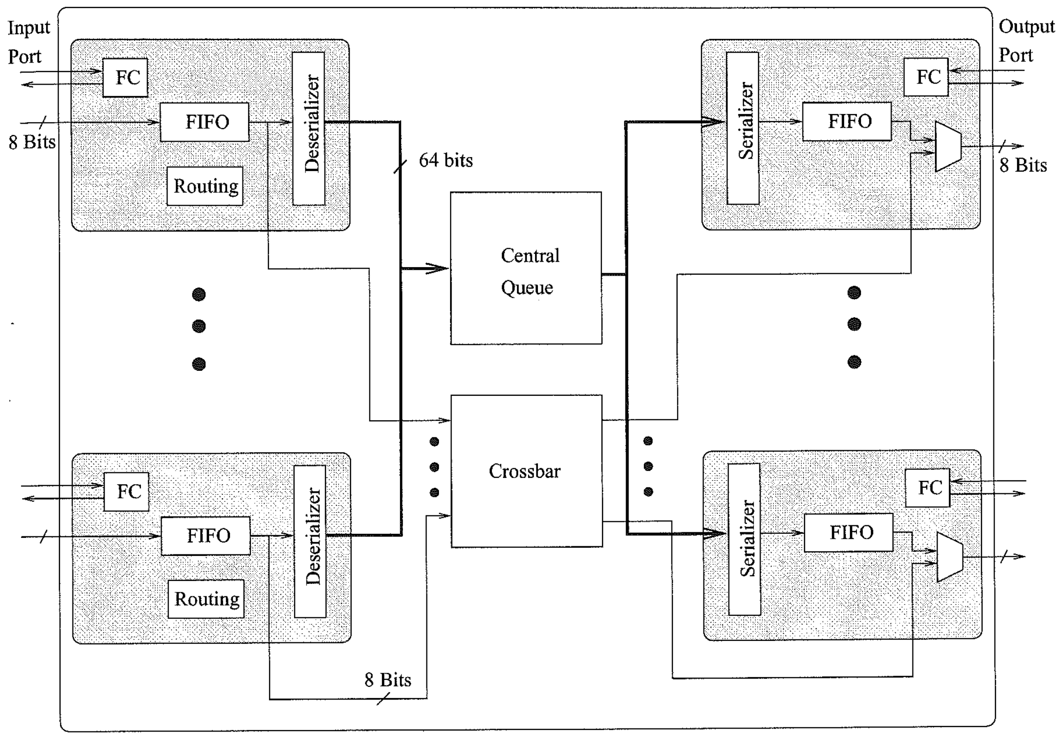


Figure 7.44. Architecture of the SP-2 router. (FC = Flow control.)

Due to propagation delays on the physical link, the input port must be able to buffer a number of flits greater than twice the channel delay: buffers are 31 flits or bytes. The transmitting side of the link consumes a token for each flit transmitted on the channel. The first flit of the message is a byte that contains the message length in flits, thus limiting message sizes to a maximum of 255 flits. Successive flits contain routing information, one routing flit for each frame (two switches) traversed by the message.

Figure 7.44 illustrates the block diagram of the architecture of the SP-2 switch. The behavior of buffered wormhole routing was described in Chapter 2. Messages are routed using source routing. Message flits are received into an input FIFO queue where the first router flit is decoded to request an output port, and CRC checks are performed. Bits 6–4 determine the output port on the first switch in a frame, while bits 2–0 determine the output port of the second switch in a frame. Bit 7 determines which of these two fields is used by the switch — when it is cleared, bits 6–4 are used. When bit 7 is set bits 2–0 are used, and the input port logic discards the routing flit and decrements the length flit in the message. Thus the message becomes smaller as it progresses toward the destination. It takes one cycle to request an output port and receive permission to transmit. If the requested port is available, it is a single cycle through the crossbar to move a flit. If the output port is not free or the request not granted by the time a chunk is received, the chunk must be buffered in the central queue. Such a chunk is referred to as a critical chunk since it is ready for immediate forwarding to an output port. Note that if any message is buffered in the central queue, a output port will not accept a request from the corresponding input port. Subsequent chunks of the message will be buffered in the central queue as space is available. Such chunks are noncritical. As the preceding chunks of a message are transmitted, noncritical chunks become critical chunks, i.e., they are ready for transmission. Storage in the central queue is allocated as follows. One chunk is statically allocated to each output port (to hold the current critical chunk). Remaining chunks are allocated dynamically to incoming messages as they become available and are stored as described in Chapter 2.

Output ports arbitrate among crossbar requests from input ports according to a least-recently-served discipline. Output port requests for the next message chunk from the central queue are also served according to a least-recently-served discipline. Note that the internal data path to the central buffer is 64 bits wide, i.e., 8 flits. Thus, one receiver can be served every cycle, with one chunk being transmitted to the central queue or the output port in one cycle. One transmitter can also be served every cycle. Each chunk is 8 flits, which takes eight cycles to receive. Thus, the bandwidth into the central queue is matched to the bandwidth out of the queue.

The SP-2 switch interface is controlled by a powerful communications coprocessor — a 64-bit Intel i860 CPU with 8 Mbytes of memory. On one side, this processor communicates with an SP-2 switch through a pair of 2-Kbyte FIFO queues. On the other side, the processor interfaces to the RS/6000 CPU Micro Channel bus via a pair of 2 Kbyte FIFO channels. One DMA engine transfers messages to and from the co-processor card and Micro Channel, and a second DMA engine transfers data between the Micro Channel FIFO buffers and the switch interface FIFO buffers. This interface can achieve bandwidths approaching that of the switch links [328].

Like the Cray T3D, the SP-2 employs source routing. The route tables are statically constructed and stored in memory to be used by the messaging software. The construction and use of the routes is performed in a manner that prevents deadlock and promotes balanced use of multiple paths between pairs of processors. In this manner an attempt is made to fully use the available wire bandwidth.

At the time of this writing the next generation of IBM switches were becoming available. The new switches utilize 16-bit flits and the crossbar path is 16 bits wide. Internally the switches operate at 75 MHz as opposed to 40 MHz in the current router. The physical channel structure remains the same, except that differential signaling results in doubling the number of signals per port to 22. The physical channel also operates at 150 MHz. Chunks are now 128 bits wide and the central queue can store 256 chunks for a total of 4 Kbytes. Since the flit size and clock speed has doubled, so have the buffers on the input ports. Flow control and buffer management necessitates 63-flit buffers, or 126 bytes. With the wider internal switch data paths and higher clock rates, the memory-to-memory bandwidth, including Micro Channel transfers, is approaching 100 Mbytes/s for large messages.

7.2.8 Network of Workstations

The first half of the 90s has seen a relentless, and large (close to doubling every 1.5 years) improvement in the price/performance ratio of workstations and personal computers. There is no evidence of this trend fading any time soon. With the concurrent trends in portable parallel programming standards, there has been an explosive growth in research and development efforts seeking to harness these commodity parts to create commercial off-the-shelf multiprocessors. A major architectural challenge in configuring such systems has been in providing the low latency and high throughput interprocessor communication traditionally afforded by multiprocessor backplanes and interconnects in a cluster of workstations/PCs. We are now seeing a migration of this interconnect technology into the cluster computing arena. This section discusses two examples of such networks.

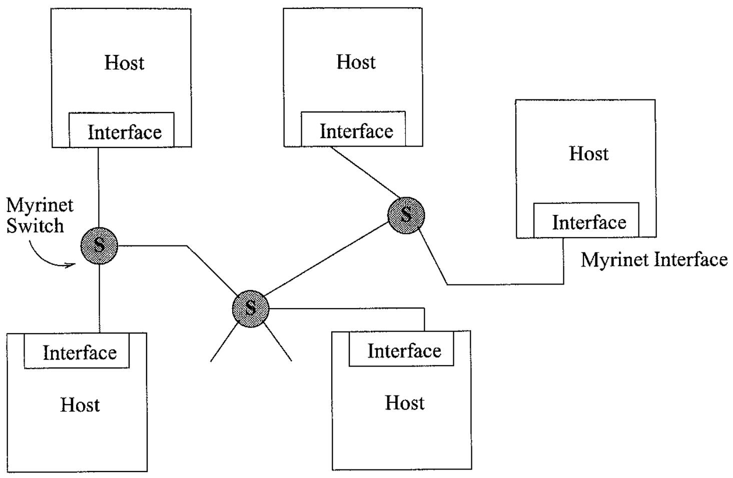


Figure 7.45. An example of a network configured with Myrinet.

Myrinet

Myrinet is a switching fabric that grew out of two previous research efforts, the CalTech Mosaic Project [314] and the USC/ISI ATOMIC LAN [107] project. The distinguishing features of Myrinet includes the support of high performance wormhole switching in arbitrary LAN topologies. A Myrinet network comprises of host interfaces and switches. The switches may be connected to other host interfaces and other switches as illustrated in Figure 7.45, leading to arbitrary topologies.

Each Myrinet physical link is a full-duplex link with 9-bit-wide channels operating at 80 MHz. The 9-bit character may be a data byte or one of several control characters used to implement the physical channel flow control protocol. Each byte transmission is individually acknowledged. However, due to the length of the cable up to 23 characters may be in transit in both directions at any given time. As in the SP-2, these delays are accommodated with sufficient buffering at the receiver. The *STOP* and *GO* control characters are generated at the receiver and multiplexed along the reverse channel (data may be flowing along this channel) to implement flow control. This slack buffer [314] is organized as shown in Figure 7.46c. As the buffer fills up from the bottom, the *STOP* control character is transmitted to the source when the buffer crosses the *STOP* line. There is enough remaining buffer space for all of the flits in transit as well as the flits that will be injected before the *STOP* character arrives at the source. Once the buffer has drained sufficiently (i.e., crosses the *GO* line) a *GO* character is transmitted and the sender resumes data flow. The placement of the *STOP* and *GO* points within the buffer are designed to prevent constant oscillation between *STOP* and *GO* states, and reduce flow control traffic. A separate control character (*GAP*) is used to signify the end-of-message. To enable detection of faulty links or deadlocked messages, non-*IDLE* characters must be periodically transmitted across each channel. Longer timeout intervals are use to detect potentially deadlocked packets. For example, this may occur over a nonfaulty link due to bit errors in the header. In this case, after 50 ms the blocked part of the packet is dropped, and a forward reset signal is generated to the receiver and the link buffers reset.

Myrinet employs wormhole switching with source routing. The message packets may be of variable length and are structured as shown in Figure 7.46c. The first few flits of the message header

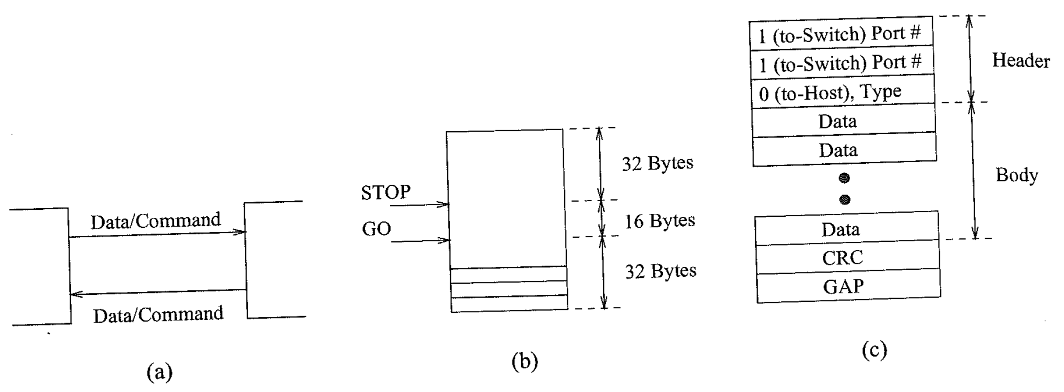


Figure 7.46. (a) The Myrinet physical link. (b) Organization of the receiver buffer. (c) Myrinet message packet format.

contain the address of the switch ports on intermediate switches. In a manner similar to the SP-2 switches, each flit addresses one of the output ports in a switch. Each switch strips off the leading flit as the message arrives and uses the contents to address an output port. Conflicts for the same output port are resolved using a recirculating token arbitration mechanism to ensure fairness. The remaining flits of the message are transmitted though the selected output port. Once the message reaches the destination, the Myrinet interface examines the leading flit of the message. This flit contains information about the contents of the packet, e.g. control information, data encoding, etc. For example, this encoding permits the message to be interpreted as an Internet Protocol (IP) packet. The most significant bit determines if the flit is to be interpreted by the host interface or a switch. The header flits are followed by a variable number of data flits, an 8-bit CRC, and the *GAP* character signifying the end of the message.

The switches themselves are based on two chips: a pipelined crossbar chip and the interface chip that implements the link-level protocols. The maximum routing latency through an eight-port switch is 550 ns [30]. At the time of this writing, four-port and eight-port switches are available with 16- and 32-port switches under development. The Myrinet host interface resembles a full-fledged processor with memory and is shown in Figure 7.47. Executing in this interface is the Myrinet control program (MCP). The interface also supports DMA access to the host memory. Given this basic infrastructure, and the programmability of the interface it is possible to conceive of a variety of implementations for the messaging layer and indeed, several have been attempted in addition to the Myrinet implementation. A generic implementation involves a host process communicating with the interface through a pair of command and acknowledgment queues. Commands to the interface cause data to be accessed from regions in memory, transferred to the interface memory, packetized, and transmitted out the physical channel. Acknowledgment of completion is by a message in the acknowledgment queue, or optionally the generation of an interrupt. Message reception follows a similar sequence of events where the header is interpreted, data transferred to locations in host memory, and an acknowledgment (or interrupt) generated. Typically a set of unswappable pages are allocated in host memory to serve as the destination and source of the interface DMA engine. In general, the Myrinet interface is capable of conducting transfers between the interface and both host and kernel memory. The MCP contains information mapping network addresses to routes, and performs the header generation. In any Myrinet network, one of the interfaces serves as the *mapper*: it is responsible for sending mapping packets to other interfaces. This map of the network is

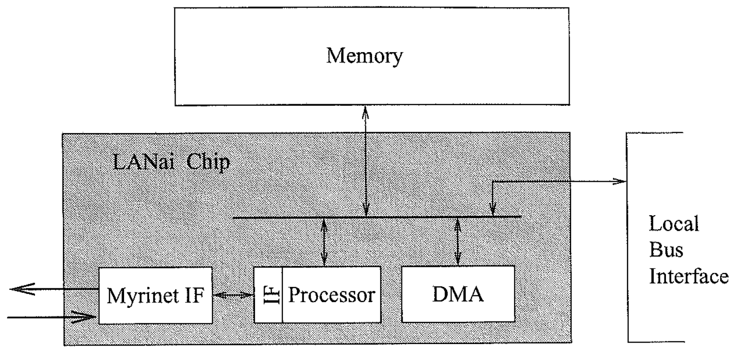


Figure 7.47. Architecture of the Myrinet interface. (DMA = Direct memory access; IF = Interface.)

distributed to all of the other interfaces. Now all routes can be created locally by each interface. The network maps are computed in a manner that is guaranteed to provide only deadlock-free routes. The attractive feature of the Myrinet network is that should the mapper interface be removed or faulty links separate the network, a new mapper is selected automatically. Periodic remapping of the network is performed to detect faulty links and switches, and distribute this information throughout the network.

More recently, Myrinet designs have evolved to a three-chip set. The single-chip crossbar is used for system area networks (SAN) where the distances are less than 3 meters. For longer distances, custom VLSI chips convert to formats suitable for longer distances of up to 8 meters typically used in LANs. A third chip provides a 32-bit FIFO interface for configurations as classical multicomputer nodes. Switches can be configured with combinations of SAN and LAN ports producing configurations with varying bisection bandwidth and power requirements. The building blocks are very flexible and permit a variety of multiprocessor configurations as well as opportunities for subsequently growing an existing network. For example, a Myrinet switch configured as a LAN switch has a cut-through latency of 300 ns while a SAN switch has a cut-through latency of 100 ns. Thus, small tightly coupled multiprocessor systems can be configured with commercial processors to be used a parallel engines. Alternatively, low latency communication can be achieved within existing workstation and PC clusters that may distributed over a larger area.

ServerNet

ServerNet [15, 156, 157, 158] is a SAN that provides the interconnection fabric for supporting interprocessor, processor-I/O and I/O-I/O communication. The goal is to provide a flexible interconnect fabric that can reliably provide scalable bandwidth. ServerNet is built on a network of switches that employ wormhole switching. Figure 7.48 illustrates the structure of the physical channel and the organization of the router. Like Myrinet, ServerNet is based on point-to-point switched networks to produce low latency communication. The structure and design of ServerNet was heavily influenced by the need to integrate I/O transfers and interprocessor communication transfers within the same network fabric.

The structure and operation of the physical channel was described in Example 7.3. The choice of 8-bit channels and command encoding was motivated by the corresponding reduction in pin-out and complexity of control logic for fault detection. The link speed of 50 MHz presents a compromise

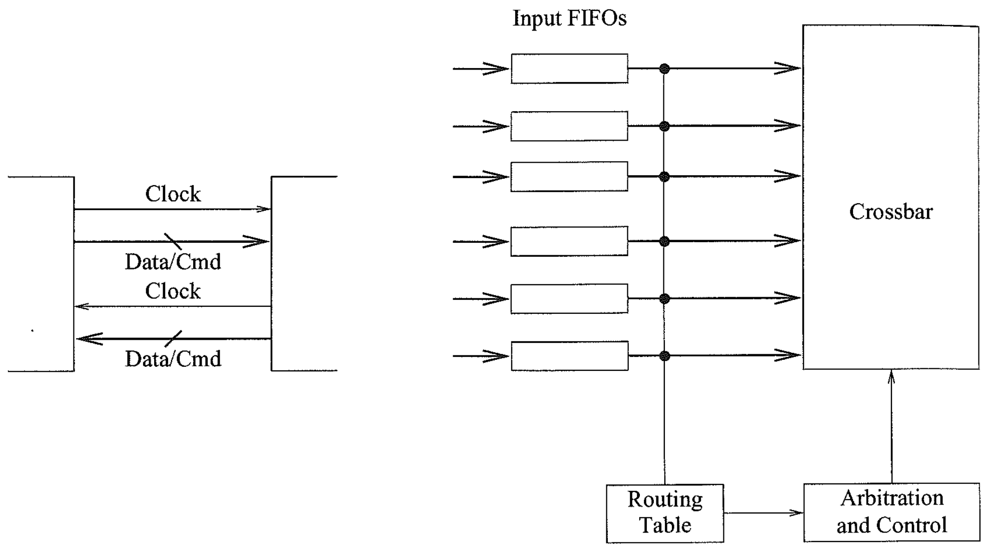


Figure 7.48. The ServerNet channel and router organization. (CMD = Command.)

between cost and the performance of modern peripheral devices. If increased bandwidth is required between end points, additional links can be added to the network. The router is a six-port, wormhole-switched router as shown in Figure 7.48. Virtual channels are not used. Since source clocking is employed to transmit flits, a synchronizing FIFO is used to receive flits at the sender's clock and output flits on the receivers clock to the input buffers. The flit buffers are 64 bytes deep. As described in Example 7.3, flow control for buffer management is implemented via commands in the reverse direction. A 1,024-entry routing table uses the destination address in the header to determine the output port. The packet now enters arbitration for the output port. Each input possesses an accumulator that determines the priority of the input channel during arbitration. The channel that gains the output channel has the associated accumulator value reduced by an amount equal to the sum of the accumulators of the remaining channels that are seeking access. The channels losing arbitration increment the value of their accumulator. This scheme avoids starvation. Deadlock freedom is achieved by design in computing the values of routing table entries. The latency through the router is on the order of 300 ns and the measured latency for a zero length packet through one switch is on the order of 3 μs.

Message headers consist of an 8-byte header which contains 20-bit source and destination IDs, message length, and additional control and transaction type information. The message header is followed by an optional 4-byte ServerNet address, variable-sized data field, and a trailing CRC. The ServerNet address is used to address a location in the destination node's address space. For interprocessor communication, the host interface will map this address to a local physical address. In this manner processor nodes can provide controlled and protected access to other nodes. All messages generate acknowledgments. Timers are used to enforce timeouts on acknowledgments. There are four message types: read request (17 bytes), read response (13 + N bytes), write request (17 + N bytes) and write response (13 bytes). The value of N represents the data size. The maximum data size is 64 bytes.

The six-port architecture can be used to create many different topologies including the traditional mesh, tree, and hypercube topologies. ServerNet is structured to use two disjoint networks: the X network and the Y network. Duplication in conjunction with acknowledgments for each packet, and error checking by means of CRC are used to realize reliable operation.

7.3 Engineering Issues

The discussions of the network optimizations and router architectures have necessarily included engineering issues and trade-offs. In this section we speculate on important trends and their impact on future networks. The design of a scalable, reliable computing system is fundamentally about the optimization of metrics such as cost, performance, and reliability in the presence of physical constraints. The emerging advances in packaging technologies, e.g., low cost MCMs, 3-D chip stacks, etc. will lead to major changes in the relationship between these physical constraints, and as a result significantly impact the cost, performance, and reliability of the next generation of interconnection networks. An accurate analysis of the effect of packaging technology and options early in the system design process will enable the implementation of systems that effectively utilize available packaging technology.

Provided with a high-level design, the fundamental question relates to where should we place die, multichip module, board, and subsystem boundaries. Die boundaries may be determined by the use of commercial parts. MCM boundaries may be determined by cost, performance, or thermal density. These partitioning decisions are critical. To make the most effective use of packaging technology we must be able to predict the impact of packaging technology on a particular design. The type of trade-offs that can be made at the packaging level include the following.

- High packaging densities (80–90%) can be realized with flip chip bonding and MCMs. This process is also more expensive and must be weighed against the reliability of a larger number of I/Os. If the implementation strategy is one of an active backplane, high-density multipath networks become viable, particularly in embedded multiprocessor applications.

- High wiring densities (50 μm pitch vs. 500 μm pitch) can be achieved in multichip packaged substrates. This has a direct, positive effect on the achievable bisection bandwidth of the network but is achieved at a higher cost and possibly entails greater power densities.

- Area array bonding can provide higher chip pin-out changing the fundamental trade-offs of what goes on- and off-chip. Off-chip speed penalties may no longer be as severe depending on the package substrate materials, e.g., organic polymer dielectric layers and copper interconnects.

- Due to the nonlinear cost/yield relationships, monolithic designs may be more cost-effectively implemented on multiple smaller die. Thus larger switches and increased buffering may become viable architectural options.

- Global interconnects on MCM substrates are not as lossy as intrachip interconnects. As a result, comparable or better performance can be obtained via the use of MCM interconnects. Furthermore, large reductions in interchip delays are possible due to lower parasitics involved in the MCM die attach technologies, hence redefining the partitioning of designs across multiple die. Several smaller die may be a cost-effective option to a larger monolithic die. This off-chip penalty has historically motivated several generations of computer architecture designs.

The above trade-offs must be made once node design and interconnect options have been proposed. These packaging technologies are making the use of multiprocessor architectures in embedded sensor processing applications quite feasible in the foreseeable future. However, doing so will require the optimization of the topologies for these new constraints.

A second issue is embedded in the advent of cluster-based computing with workstations and PCs. We are seeing the migration of the expertise gained from the design of several generations of high speed multiprocessor interconnects, to the design of low-latency, robust cluster interconnects. Among the many architectural issues being resolved in this context is the point at which the network is integrated into the local node (e.g., memory, as an I/O device, etc.), software/hardware interface to network bandwidth, and the design of message layers that realize low-latency communication when constrained to use commodity parts. Some of the software issues are discussed in Chapter 8.

7.4 Commented References

Dally [71] originally studied the design of k-ary n-cube networks under the assumption of VLSI implementations wherein network implementations were wire-limited. Wiring constraints were captured in the form of bisection width constraints. Networks were compared by fixing the available bisection width to that of a binary hypercube with bit-wide, full-duplex, data channels. Under different wire delay models the analysis concluded that low-dimensional networks provided a better fit between form and function, resulting in better overall latency and throughput characteristics. Subsequently, Abraham and Padmanabhan [1] studied the problem under the assumption of fixed pin-out constraints. In this environment, the wider channel widths of low-dimensional networks were offset by the greater distances. Thus, their analysis tended to favor higher-dimensional networks. Bolding and Konstantinidou [33] observed that the maximum available pin-out is not independent of the bisection width. Thus, they proposed using the product of the bisection width and pin-out as a measure of the cost of the network. Thus comparisons could be made between two (quantifiable) equal-cost networks. Agarwal's [3] analysis included the effect of switch delays, which was missing from prior analysis. In this model, the analysis favored slightly higher-dimensional networks than Dally's analysis [71]. One of the principal inhibitors of the use of higher-dimensional networks has been the wire delay due to the long wires that result when these networks are laid out in the plane. Scott and Goodman [310] studied the ability to pipeline bits on the wire, effectively using these long propagation delays as storage to create pipelined wires. As result, their analysis too favored higher-dimensional networks. These techniques are reflected in the switching fabric that is produced in current generation networks. Finally, the continuous evolution of high-density packaging with area-bonded die is resulting in the capability for producing chips with very high number of I/O connections. This relaxation of the pin constraint will no doubt favor lower-dimensional networks due to the constraints on bisection width.

The first routers were packet-switched and circuit-switched, following from previous experiences with multicomputer communications in the LAN environment. While packet-switched networks enabled deadlock-free routing via structured memory allocation, message latencies in tightly coupled multiprocessors were very high due to the linear relationship between interprocessor distance and latency. The advent of cut-through routing significantly reduced this latency, but the need to buffer packets at the nodes required the involvement of the node processors: we can expect this trend to be quite inefficient in the tightly coupled multiprocessor world. The advent of small-buffer cut-through or wormhole switching made it feasible to construct single-chip, compact, fast routers that did not require the services of intermediate nodes. A generation of router chips were based on this approach [77, 79, 255]. As densities continued to increase, it was possible to include enough buffer space on the single chip to support VCT switching [41, 83, 188]. With better understanding of the advantages as well as limitations of cut-through switching, efforts started focusing on optimizations to further improve the performance or reliability leading to new router architectures [7, 316, 327].

Finally, we are seeing a convergence of techniques for routing in LANs and multiprocessor backplanes. As clusters of workstations become economically viable, focus has shifted to the support of efficient messaging mechanisms and hardware support. The new generation of router architectures reflects this trend [30, 156]. We can expect this trend to continue, further blurring the distinctions between clusters of workstations and tightly coupled multiprocessor backplanes.

EXERCISES

7.1 Rewrite the expression for the no-load message latency in a k-ary n-ary cube under the following assumptions. Rather than a router architecture that is buffered at the input and output, assume that the router has input buffering only. Thus, in a single cycle, a flit can progress from an input buffer on one router to an input buffer of another router, if the corresponding output link at the first router is free. Furthermore, assume that the routing delay and the switch delay scale logarithmically with the network dimension.

Solution We have the equation for the no-load latency given by

$$t_{wormhole} = n\frac{k}{4}\left[r + s + 1 + \left(\frac{n}{2} - 1\right)\log_e k\right] + \max\left[s, 1 + \left(\frac{n}{2} - 1\right)\log_e k\right]\left\lceil\frac{L}{W}\right\rceil \qquad (7.25)$$

Each stage of the message pipeline from the source to the destination now consists of transmission across the router and the link. Therefore, the coefficient of the second term becomes the sum of the wire delay and switch delay. The new expression is

$$t_{wormhole} = n\frac{k}{4}\left[r + s + 1 + \left(\frac{n}{2} - 1\right)\log_e k\right] + \left[s + 1 + \left(\frac{n}{2} - 1\right)\log_e k\right]\left\lceil\frac{L}{W}\right\rceil \qquad (7.26)$$

This equation can be rewritten as

$$t_{wormhole} = rn\frac{k}{4} + \left[s + 1 + \left(\frac{n}{2} - 1\right)\log_e k\right]\left(n\frac{k}{4} + \left\lceil\frac{L}{W}\right\rceil\right) \qquad (7.27)$$

The first term is the component due to routing delays (in the absence of contention). The coefficient of the second term corresponds to the clock cycle time of the network.

7.2 Individual analysis of the networks has been based on their use of a single resource, e.g., wiring density or pin-out. Actually both node degree and bisection are related. A better measure of network cost/performance appears to be the product of node degree and bisection width proposed in [33]. Determine the relationship between channel widths of two networks of equal cost using this performance metric. Compare a binary hypercube and an equivalent-sized 2-D mesh using this metric.

Solution In general, consider two topologies of dimension r and s with the same number of nodes. We know that the product of their bisection bandwidth and pin count are equal. The width of each dimension in the r-dimensional topology is given by $N^{\frac{1}{r}}$. This formula is derived from the relation $N = k_m^r$. The pin-out is given by (including only data pins) $2rw_r$ where w_r is the channel width. We can now write the products of the bisection bandwidth and pin-out as

$$(2w_rN^{1-\frac{1}{r}})(2rw_r) = (2w_sN^{1-\frac{1}{s}})(2sw_s)$$

From the above expression we have

$$\frac{w_r}{w_s} = \sqrt{\frac{sN^{1-\frac{1}{s}}}{rN^{1-\frac{1}{r}}}}$$

Substituting the correct number of dimensions for a binary hypercube and a 2-D mesh, we have

$$\frac{w_{hyp}}{w_{mesh}} = \sqrt{\frac{2N^{1-\frac{1}{2}}}{rN^{1-\frac{1}{r}}}}$$

which simplifies to

$$\frac{w_{hyp}}{w_{mesh}} = \sqrt{\frac{4\sqrt{N}}{N\log_2 N}}$$

7.3 Early in this chapter, an expression was provided for the relative length of the longest wire in a k-ary n-cube when embedded in two dimensions. Repeat this analysis when a k-ary n-cube is embedded in three dimensions.

Solution Consider the base case of of a 3-D cube with each dimension interleaved as shown in Figure 7.2. A k-ary n-cube is embedded in three dimensions by embedding $\frac{n}{3}$ dimensions of the network into each of the three physical dimensions. Let us assume that the number of dimensions is a multiple of 3. Each additional dimension beyond the first three increases the number of nodes in the network by a factor of k. Embedding in three dimensions provides an increase in the number nodes in each physical dimension by a factor of $k^{\frac{1}{3}}$. If we ignore wire width and wiring density effects, the length of the longest wire in each physical dimension is also increased by a factor of $k^{\frac{1}{3}}$. Therefore for n dimensions the longest wire will grow by a factor of $k^{\frac{n-3}{3}} = k^{\frac{n}{3}-1}$. Note that similar arguments lead to the general case of the wire length of embeddings in d dimensions being $k^{\frac{n}{d}-1}$.

PROBLEMS

7.1 Cascading is employed to configure a 512-node IBM SP-2 system. How many disjoint paths exist between any pair of processors?

7.2 Flow control operations across a link take a finite amount of time. A message may be in transit during this period. In pipelined links, several messages may be in transit. The receive buffer size must be large enough to store the phits in progress over a link. Given the maximum number of phits in transit on a physical channel, write an expression for the minimum receive buffer size in phits to prevent any loss of data.

7.3 Consider an m-dimensional and n-dimensional tori of equal cost, where cost is given by the product of the bisection width and pin-out that utilizes this bisection width. Write analytical expressions for the relationships between the channel widths of the two networks and the bisection bandwidths of the two networks.

7.4 Consider embedding a k-ary n-cube in three dimensions rather than two dimensions. Rewrite the expression for average message latency by modifying the expressions that capture the effect of wire length. Compare this to the expression derived in this chapter.

7.5 We are interested in determining the maximum message injection rate for a given bisection bandwidth. Assume we have a 16×16 torus with 1-flit-wide, half-duplex channels. Message destinations are uniformly distributed. Find the maximum message injection rate in flits/node/cycle.

Hint Every message generated by a node is equally likely to cross the bisection, or be addressed to a node in the same half of the network.

7.6 A $4 \times 4 \times 8$ Cray T3D system experiences a single node failure. There are no spare nodes. All of the routing tables are updated to use the longer path in the dimension. Now we are concerned about deadlock. Is deadlock possible? Justify your answer with a proof or an example.

7.7 Source routing as described in the Cray T3D can cause imbalance in the use of virtual channels. For an eight-node ring, compute the degree of imbalance across each physical link. The degree of imbalance across a physical link is defined as the ratio of (1) the difference in the number of source-destination pairs that use virtual channel 0 and virtual channel 1 across the link, and (2) the total number of source-destination pairs that use the link. The choice of the dateline link does not matter.

7.8 What is the bisection bandwidth in bytes of the following machine configurations?

1. A $4 \times 8 \times 4$ Cray T3D
2. An 8×14 Intel Paragon
3. A 64-node IBM SP-2

7.9 Using the following values for the constants in the parameterized cost model shown in Table 7.4, compute the routing latency and the maximum flit bandwidth available to a message.

Constant	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9
Value (ns)	0.4	0.6	2.2	2.7	0.6	0.6	1.4	0.6	1.24	0.6